Introduction

A la fin de ce chapitre, vous serez capables :

- d'expliquer pourquoi des langages dédiés à la résolution de problèmes mathématique ont été crées.
- de citer quelques outils et langages populaire dans ce domaine
- de dire comment choisir un langage
- d'expliquer pourquoi vous allez apprendre Octave

Pourquoi des langages dédiés au calcul mathématique ?

La réponse à cette question est simple : c'est pour éviter aux programmeurs de fournir trop d'efforts sur les aspects techniques de programmation leur permettant ainsi de se focaliser sur les problèmes mathématiques. Le premier langage de programmation qu'on pourrait qualifier d'évolué a été crée pour permettre de transcrire des formules mathématiques. C'est le langage FORTRAN (FORmula TRANstalor). Cependant ce langage, à l'instar d'autres langages à usage général comme Algol, C ou Pascal, n'est pas proche du langage mathématique. C'est pour cette raison que des langages encore plus évolués et dédiés aux mathématiques ont été crées.

Langages et outils populaires

Dans le domaine scientifique, il existe un éventail de langages aussi diversifié que l'est le domaine des mathématiques : Analyse de données, statistiques, algèbre, analyse numérique, traitement d'images, traitement du signal, etc. Ainsi, on peut citer R, Gnumeric, Excel, SPSS et toute une panoplie de tableurs dédiés principalement aux traitements statistiques et l'analyse de données. Dans le domaine du calcul vectoriel et matriciel, de l'algèbre linéaire, du traitement d'image, du calcul intégral, de la résolution des systèmes d'équations, on peut citer : MATLAB, Scala, PyLab (Python), Octave et Scilab.

Choisir un langage

La liste de tous les langages de programmation est longue (plus de 1000 actuellement). Faire un choix n'est souvent pas évident. La question fondamentale que vous devez vous poser pour est : « *qu'elle est le type de problème que je veux résoudre ?* ». C'est ce qui va vous orienter vers tel ou tel autre groupes de langages. En effet, il existe des langages à usage général comme Python, C, Pascal, C++, Java et il existe des langages plutôt spécialisés comme : Matlab, R, Octave, Scilab, Javascipt, HTML, SQL, PHP, etc.

Dans le domaine du calcul mathématiques la liste de ces langages se réduit, mais reste assez diversifiée : TkSolver, Matlab, Scilab, sageMath, Analytica, FreeMat, et bien d'autres encore !

Par ailleurs, d'autres critères peuvent être pris en compte dans le choix d'un langage dédiés aux mathématiques. Par exemple, si les problèmes que vous avez à résoudre exigent du temps réel ou nécessitent une haute performance dans les calculs, des langages compilés sont favorisés par rapport aux interpréteurs.

Remarque :

- Un langage compilé exige l'écriture du programme complet pour pouvoir générer un exécutable. Le point fort de ces langages réside dans le fait que les exécutables qu'ils génèrent sont optimisés pour les processeurs qu'ils ciblent. Leur point faible réside dans le fait que ces exécutables ne sont pas portables d'un type de processeur vers un autre. C, Pascal, Fortran en sont des exemples.
- Un langage interprété permet d'exécuter les instructions (commandes) du programmeur au fur et à mesure de leurs saisies. Ils sont réputés plus lents que les langages compilés, mais les programmes qu'ils génèrent sont portables d'une machine à une autre pourvue que ces machines disposent d'un interpréteur.

Si vous débuter, la facilité de prise en main du langage est un critère non négligeable. Ainsi, favorisez les langages de haut niveau comme Matlab, Octave ou Python par rapport à Fortran, Java, C ou C++.

Enfin, d'autres critères comme la popularité, la pénétration du langage dans le monde économique (industrie, entreprises), les bibliothèques disponibles et la dynamique d'une communauté de développeurs doivent être pris en compte.

Pourquoi étudier Octave ?

Matlab est considéré, par beaucoup de programmeurs, comme la référence dans le domaine de la programmation scientifique. En effet, l'Institute of Electrical and Electronics Engineers (IEEE), dans son rapport annuel sur les meilleurs langages de programmation pour l'année 2016, le classe dans le Top 10 des meilleurs langages pour le développement d'applications d'entreprise, de bureau et d'applications scientifiques. Il est devancé uniquement par des langages à usage général comme Java et Python et un seul langage spécialisé dans le domaine des statistiques qui est R.

Dans ce cours, on s'intéresse à un langage plutôt dédié aux mathématiques, au sens général, et non pas uniquement aux statistiques. C'est pour cette raison que nous n'allons pas nous intéresser à R. Il nous reste donc Matlab, mais celui-ci n'est pas gratuit. Heureusement, il existe plusieurs alternatives gratuites à ce langage. Scilab, Octave, Freemat, Rlab, et LabView en sont des exemples.

Afin de profiter du langage Matlab gratuitement, nous allons présenter, dans ce cours, le langage Octave qui est le clone le plus proche du logiciel Matlab.

QCM0 - Introduction

Q1 : Des langages dédiés à la résolution de problèmes scientifiques (calcul numérique, statistiques, simulation, etc.) ont été crées parce que :

- ☐ Les anciens langages comme Fortran, Pascal et C ne sont pas puissants.
- ☐ Les anciens langages comme Fortran, Pascal et C ne sont pas rapides.
- ☐ Les anciens langages comme Fortran, Pascal et C ne sont pas proche des formulations mathématiques couramment utilisées.

Q2 : Des langages spécialisés dans le domaine des mathématiques (au sens large) sont populaires. En voici des exemples :

- ☐ Tableurs comme Excel et Gnumeric
- ☐ Interpréteurs comme R, Matlab, Scilab
- ☐ Interpréteurs comme Python
- ☐ Compilateurs comme C et Pascal

Q3 : Des langages interprétés génèrent des programmes plus rapides que ceux générés par des compilateurs :

- ☐ Vrai
- ☐ Faux

Q4 : Un interpréteur est un logiciel permettant d'exécuter des commandes (en respectant un langage de programmation) au fur et à mesure de leurs saisies :

- ☐ Vrai
- ☐ Faux

Q5 : Un compilateur est un logiciel permettant d'exécuter des commandes (en respectant un langage de programmation) au fur et à mesure de leurs saisies :

- ☐ Vrai
- ☐ Faux

Q6 : Dans le domaine du calcul hautes-performances comme les prévisions météorologiques, on utilise des langages interprétés :

- ☐ Vrai
- ☐ Faux

Q7 : Dans le domaine du calcul hautes-performances comme les prévisions météorologiques, on utilise des langages compilés :

- ☐ Vrai
- ☐ Faux

Q8 : Si je veux écrire un programme pour faire la simulation des mouvements des planètes dans une galaxie (demandant des traitements intensifs), je privilégie Fortran à Octave :

- ☐ Vrai
- ☐ Faux

Q9 : Si je veux écrire un programme pour faire du calcul statistique, je privilégie R à Matlab :

- ☐ Oui
- ☐ Non

Q10 : Si je veux écrire un programme pour faire du calcul vectoriel et matriciel, je privilégie R à Octave :

- ☐ Oui
- ☐ Non

Q11 : Matlab est un logiciel gratuit utilisé à des fins de calcul numérique et permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des scripts et dispose d'un riche *toolbox* permettant d'étendre ses possibilités :

- ☐ Oui
- ☐ Non

Q12 : Octave est un logiciel gratuit et open source utilisé à des fins de calcul numérique et permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des scripts et dispose d'un ensemble de packages (bibliothèque) permettant d'étendre ses possibilités (statistiques, traitement du signal, gestion de base de données, etc.) :

- ☐ Oui
- ☐ Non

Q13 : Fortran, C et Pascal sont plus faciles à utiliser que Matlab, Octave, Scilab et Python :

- ☐ Oui
- ☐ Non

Q14 : Python est un langage à usage général, mais dispose de module spécialisés en mathématique, ce qui fait de lui un sérieux concurrent à Matlab :

- ☐ Oui
- ☐ Non

Chapitre 1 –Prise en main d'Octave

A la fin de ce chapitre, vous serez capables :

- D'installer octave
- D'expliquer son interface graphique
- D'utiliser ses différentes fenêtres
- Commande
- Historique des commandes
- Espace de travail
- Editeur de script
- Documentation
- D'expliquer comment installer des packages

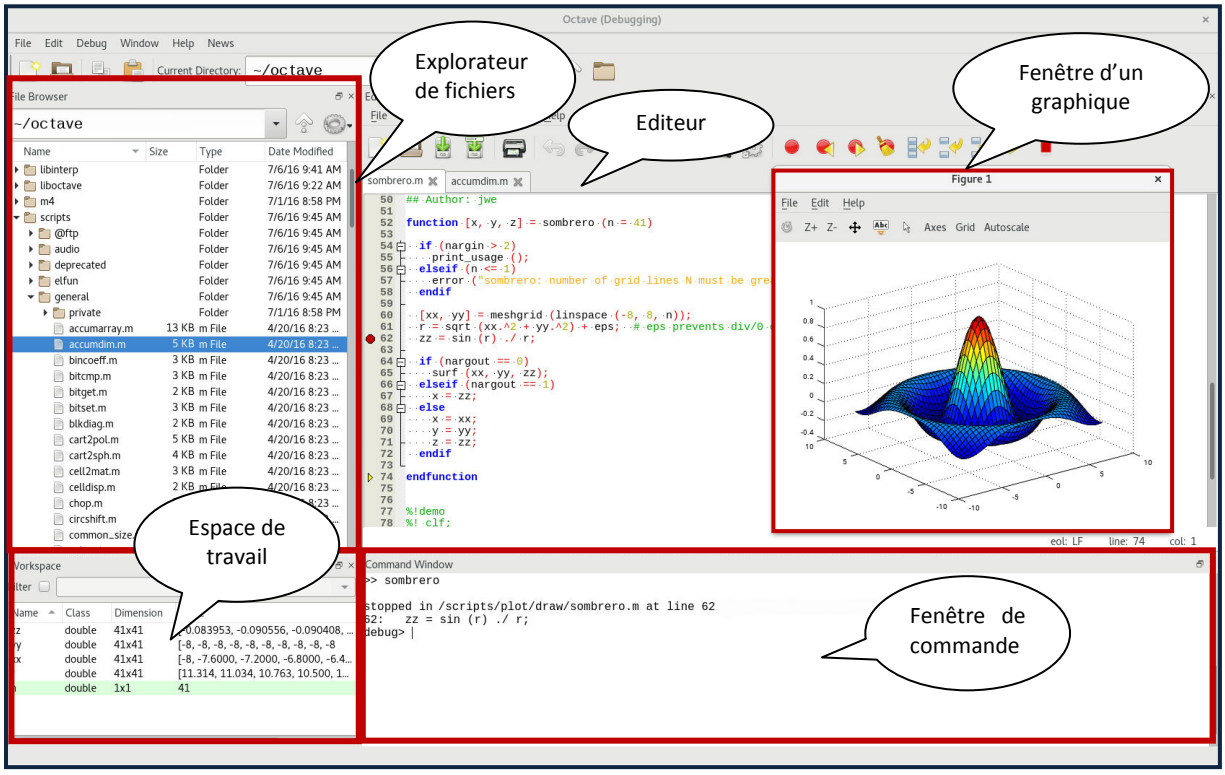
I.1 – Installation

Octave est gratuit open source et multiplateformes. Vous pouvez vous le procurer en visitant son site web officiel : http://www.octave.org. Son installation est très simple. Sous Linux, vous suivez la procédure habituelle d'installation de logiciel en utilisant un gestionnaire de paquetage de votre distribution (***apt*** pour *Debian* et ***yum*** pour *Redhat* et *Fedora*. Par exemple, sous *Ubuntu* (*Debian*), il suffit de taper au terminal la commande suivante : « **sudo apt-get install octave** ». Vous pouvez bien évidemment utiliser un gestionnaire graphique comme *Synaptic* ou la logithèque *d'Ubuntu* pour faire la même chose.

Sous Windows, le processus n'est pas plus compliqué. Il suffit de télécharger la le code exécutable de la dernière version d'Octave depuis sont site web officiel, puis d'exécuter ce fichier et de suivre la procédure d'installation en faisant quelques configurations mineures comme indiquer le lieu (dossier du disque) où vous souhaiter installer Octave.

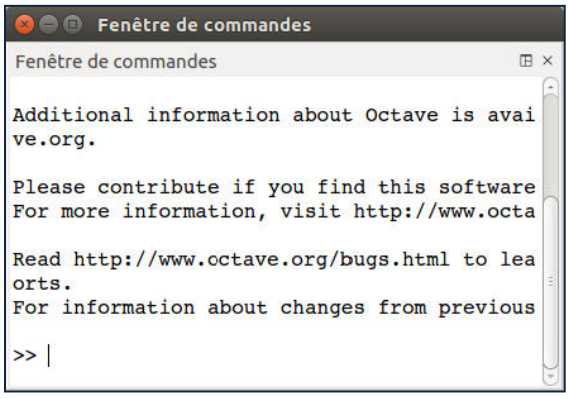
I.2 – Interface graphique

Les anciennes versions d'Octave s'utilisent en mode commande à travers un *shell*. Les dernières versions offrent un environnement graphique plus agréable, semblable à celui de Matlab. Ainsi, lorsque vous le lancer, une interface graphique s'affiche. Elle est composée de plusieurs fenêtres : Commandes, historique des commandes, explorateur de fichier, éditeur, espace de travail et documentation. Il est aussi possible d'ouvrir des fenêtres de graphiques.



I.3 - Utiliser les fenêtres d'Octave

Fenêtre de commande :

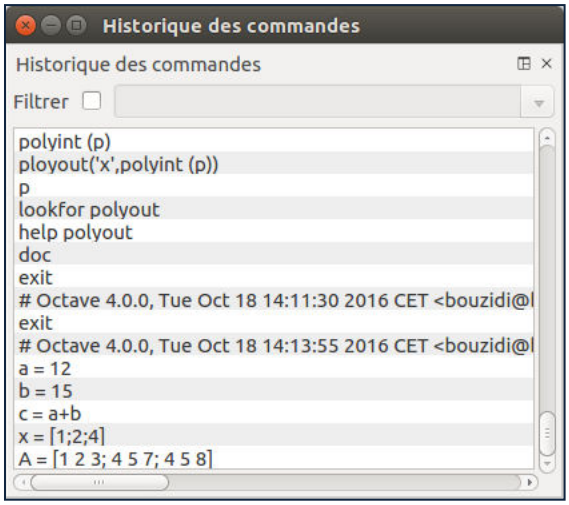


C'est le lieu où a lieu le maximum d'interaction avec vous étant donné que c'est dans cette fenêtre que vous allez introduire les commandes Octave. C'est où vous allez dire à Octave de créer des variables et d'exécuter des fonctions et des scripts.

Octave s'occupera à répondre à chacune des commandes que vous tapez !

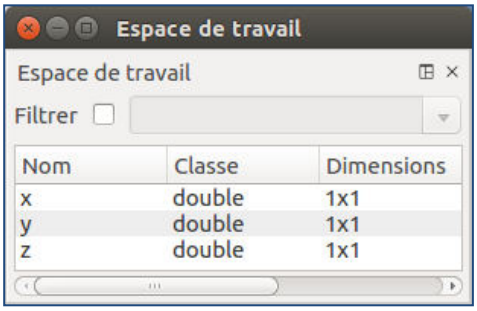
Notez bien que vous pouvez effacer le contenu de la fenêtre de commande en tapant la commande « **clc** »

Historique des commandes :



Cette fenêtre affiche, dans l'ordre, toutes les commandes que vous avez tapez. Notez-bien que vous pouvez gérer à votre guise l'historique de vous commandes. En particulier, vous pouvez l'effacer, le sauver sur disque, l'éditer ou l'exécuter totalement ou partiellement. C'est la commande « ***history*** » qui vous permettra cela. Je vous invite à tapez « ***help history*** » pour découvrir les différentes options cette commande et les possibilités qu'elle offre.

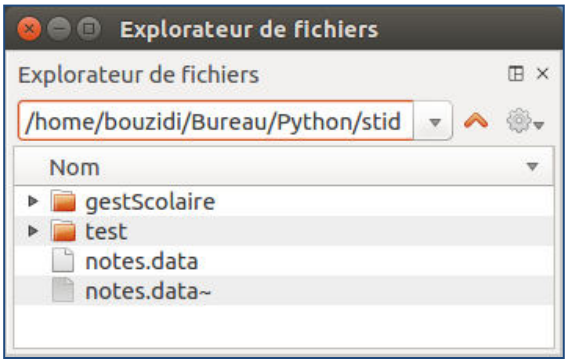
Espace de travail :



Dans cette fenêtre sont affichées toutes les variables et les fonctions que vous créez. Elle vous indique les noms des variables, leur type, leurs dimensions et les valeurs qu'elles contiennent.

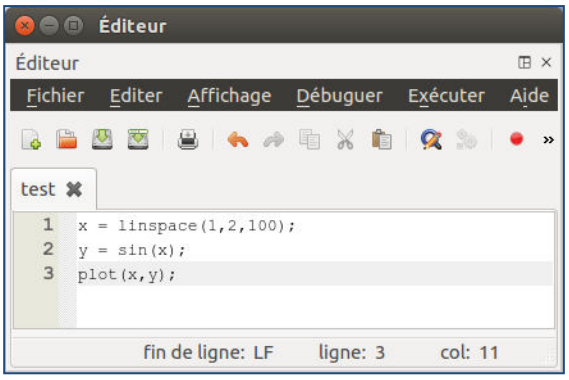
Notez-bien que vous pouvez supprimer des variables à l'aide de la commande « ***clear*** ».

Explorateur de fichiers :



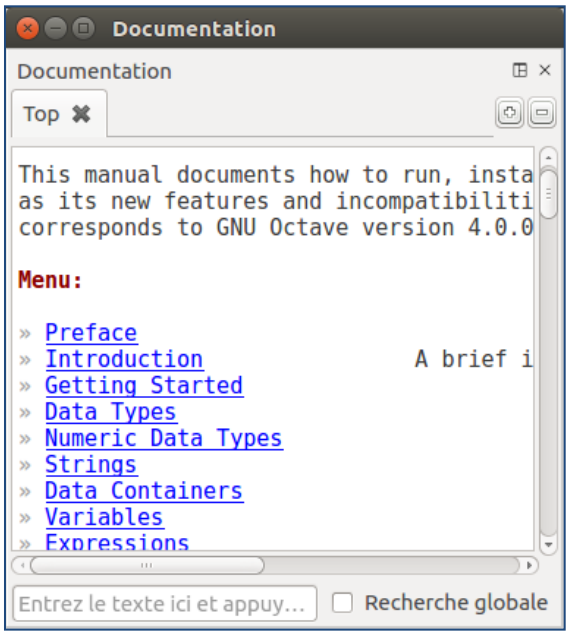
Comme son l'indique, cette fenêtre vous permet de naviguer sur vous supports de stockage (disque, CD-ROM, clés USB, etc.). Elle est composée de deux zones : une zone indiquant le dossier courant et une zone plus étendue indique le contenu du dossier courant. Cette fenêtre vous permet à tous moment de changer votre dossier courant.

Editeur :



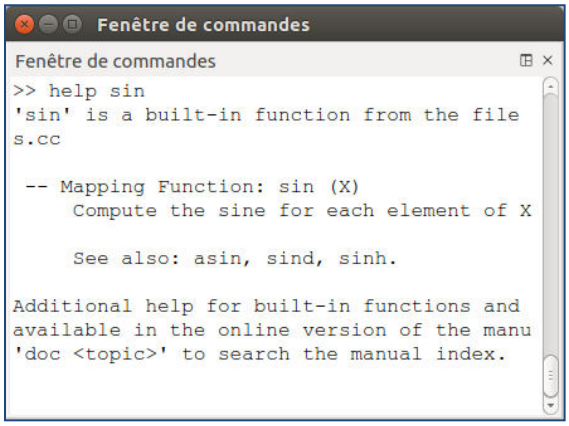
Il s'agit d'un éditeur de textes doté de fonctionnalités vous permettant d'éditer, de sauver, de mettre au point et d'exécuter vos scripts.

Documentation :



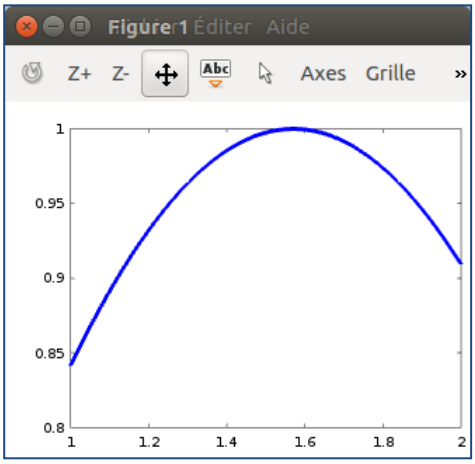
Cette fenêtre vous donne accès à une documentation détaillée sur Octave. Vous pouvez naviguer à travers une barre de menu principal et à chaque fois que vous cliquez sur un menu, une nouvelle page s'affiche.

Notez bien, qu'il existe une autre manière d'avoir de l'aide. Il s'agit de la commande « help ». En effet, dès que vous voulez savoir comment utiliser une fonction ou ce qu'elle fait, il suffit de tapez la comme « help » suivi du nom de votre fonction.



Dans l'exemple ci-dessus, j'ai demandé de l'aide sur la fonction « ***sin*** ».

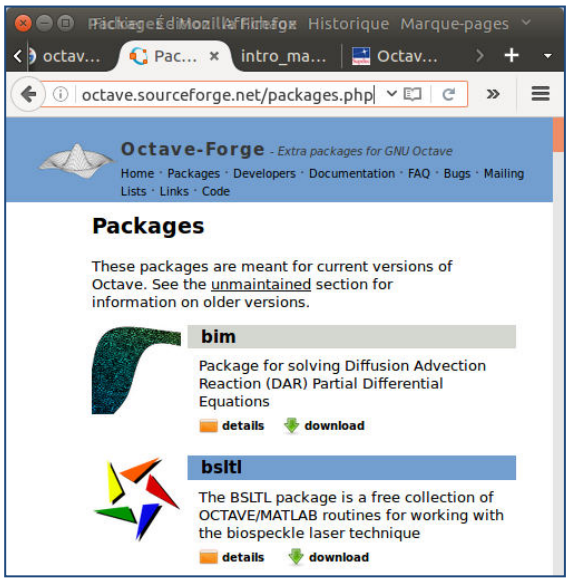
Graphiques :



La figure ci-dessus a été affichée grâce la commande « plot » que nous verrons dans le chapitre sur les graphiques.

I.4 – Installer de nouveaux packages

Matlab dispose d'un *toolbox* très riche permettant d'étendre ses fonctionnalités et ses possibilités. Octave dispose de l'équivalent de cette *toolbox* qu'on appelle « packages ». Ils sont disponibles en téléchargement depuis le site officiel d'Octave :



Un package est une sorte de bibliothèque renfermant des fonctions. Vous disposez d'une commande vous permettant de gérer les packages : c'est « ***pkg*** ». Les options « *install* », « *update* », « *uninstall* » et « *load* » rajouter à la commande « *pkg* » permettant respectivement d'installer, de mettre à jour, de charger et de désinstaller un package.

QCM 1 – Octave : Prise en main

Q1 : Pour installer « **octave** », sous Windows, j'ouvre une console DOS (terminal) et je tape la commande suivante : « **sudo apt-get install octave** » :

☐ Vrai ☐ Faux

Q2 : Pour installer « **octave** », sous Ubuntu, j'ouvre une console DOS (terminal) et je tape la commande suivante : « **sudo apt-get install octave** » :

☐ Vrai ☐ Faux

Q3 : Pour installer « **octave** », sous Ubuntu, je peux utiliser le terminal (console) en mode commande suivante : « **sudo apt-get install octave** ». Je peux aussi utiliser, en mode graphique, la logithèque d'Ubuntu :

☐ Vrai ☐ Faux

Q4 : **Octave** est multiplateforme je peux l'installer sous Windows, Linux, McOS et même Android !

☐ Vrai ☐ Faux

Q5 : Le site officiel d'**Octave** est :

- ☐ http://www.octave.org
- ☐ http://www.octave.fr
- ☐ http://www.apprendre-octave.org

Q6 : Lorsque vous lancez **Octave** une interface graphique composée des fenêtres suivantes s'affiche :

- ☐ Explorateur de base de données
- ☐ Explorateur de fichier
- ☐ Explorateur de données
- ☐ Espace de travail
- ☐ Fenêtre de commande
- ☐ Editeur
- ☐ Documentation
- ☐ Aide
- ☐ Historique des commandes
- ☐ Historique des scripts
- ☐ Historique des fonctions
- ☐ Packages

Q7 : La fenêtre de commande vous permet de saisir des scripts :

☐ Vrai ☐ Faux

Q8 : L'éditeur vous permet de saisir des scripts de les mettre au point et de les exécuter:

☐ Vrai ☐ Faux

Q9 : Dans la fenêtre de commande je peux saisir des instructions qui s'exécutent juste après avoir tapé la touche « ENTER »:

☐ Vrai ☐ Faux

Q10 : Si je veux avoir de l'aide sur la fonction « ***plot*** », je saisi, dans la fenêtre de commande, ceci :

- ☐ aide plot
- ☐ help plot
- ☐ help plot()
- ☐ Help plot
- ☐ Doc plot()

Q11 : Si je veux effacer le contenu de la fenêtre de commande (attention à ne pas confondre avec l'espace de travail qui renferme toutes les variables de ma session de travail), je saisi, dans la fenêtre de commande, ceci :

- ☐ clear
- ☐ effacer
- ☐ clc

Q12 : Si je veux avoir de l'aide pour comprendre la commande « ***history*** », je saisi, dans la fenêtre de commande, ceci :

- ☐ aide history
- ☐ Help history
- ☐ help history

Q13 : Je suppose que vous savez demander à Octave de vous expliquer comment utiliser la commande « ***history*** ». Je vous demande de me donner la commande permettant :

A - d'afficher les 10 dernières lignes de l'historique des commandes:

.....

B – de sauver sur disque dans le fichier « historique » l'historique des commandes:

.....

C – d'effacer l'historique des commandes:

.....

D – de lire l'historique des commandes depuis le disque à partir du fichier « *historique* »:

.....

TP1 – Octave : Prise en main

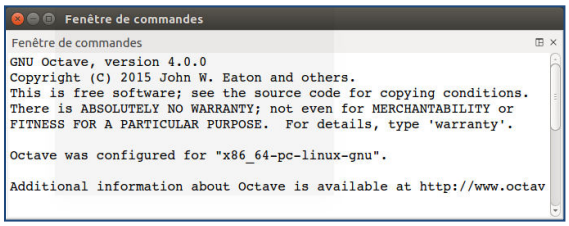
A la fin de ce TP, vous serez capables :

- D'identifier les différentes fenêtres et quelques éléments visuels d'octave et d'indiquer leurs rôles
- D'organiser votre travail
- D'interagir à travers la fenêtre de commande
- De gérer l'historique des commandes
- De demander de l'aide et de la documentation
- De gérer les packages

1 - Identifier les différentes fenêtres d'Octave

Je vous invite à lancer le logiciel « **Octave** ». Vous pouvez le faire à travers le terminal de Linux (console DOS de Windows) en tapant « **octave** », ou vous servir du menu « démarrer » de Windows. Une fois lancé, remarquez que ce logiciel présente un environnement graphique composé de plusieurs fenêtres. Je vous invite à découvrir cet environnement graphique.

1.1 - Indiquez à quoi servent les fenêtres suivantes :

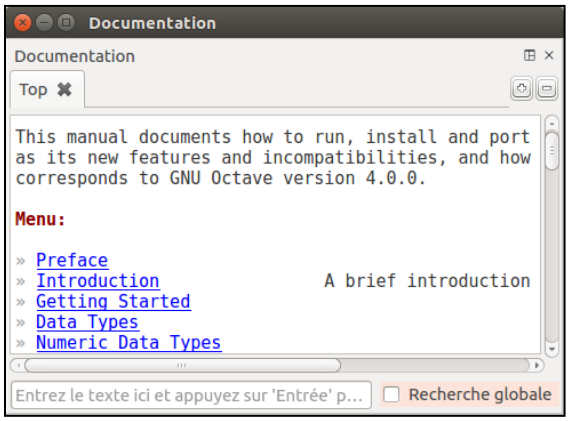


.....

.....

.....

.....

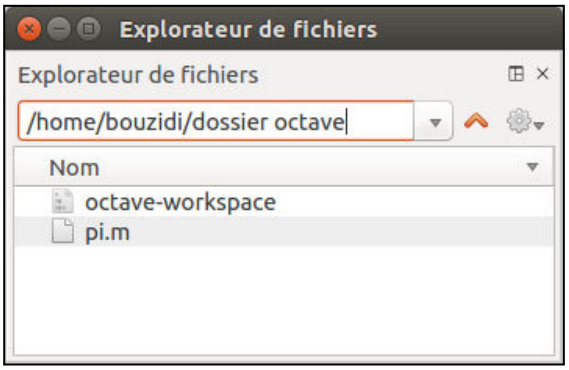


.....

.....

.....

.....



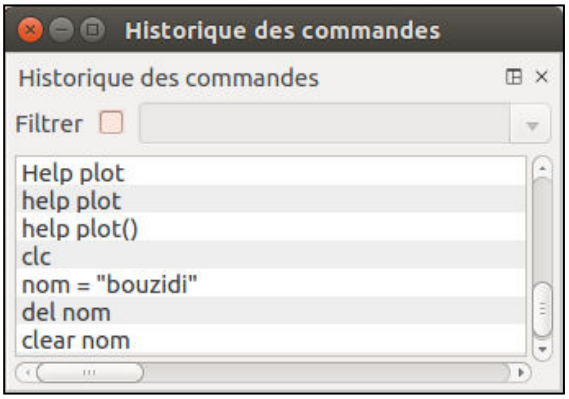
.....

.....

.....

.....

.....



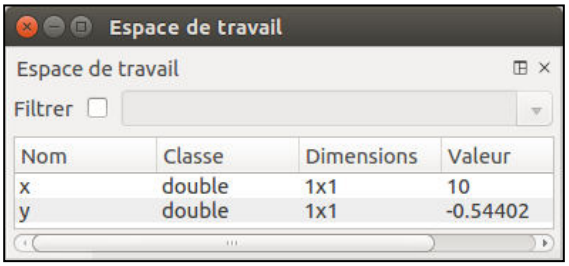
.....

.....

.....

.....

.....

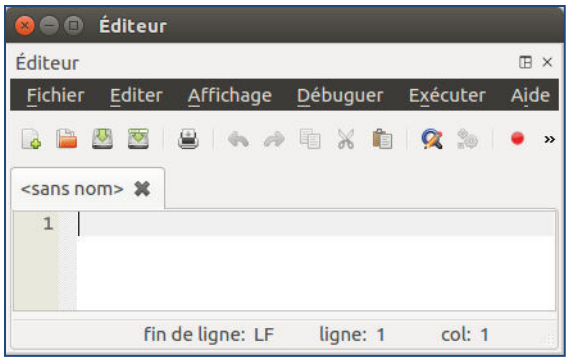


.....

.....

.....

.....



.....

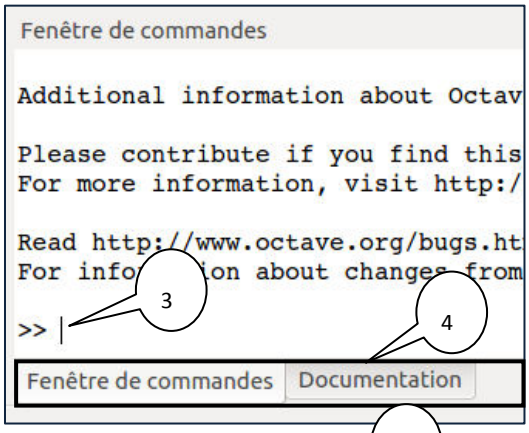
.....

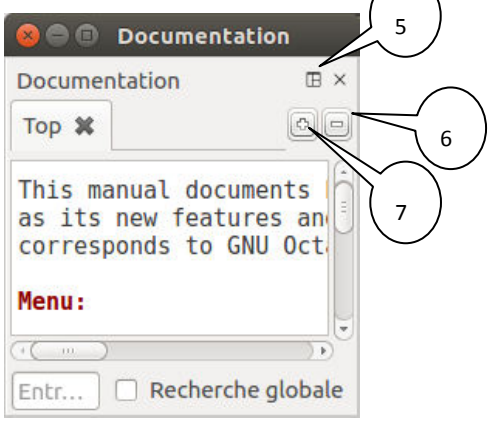
.....

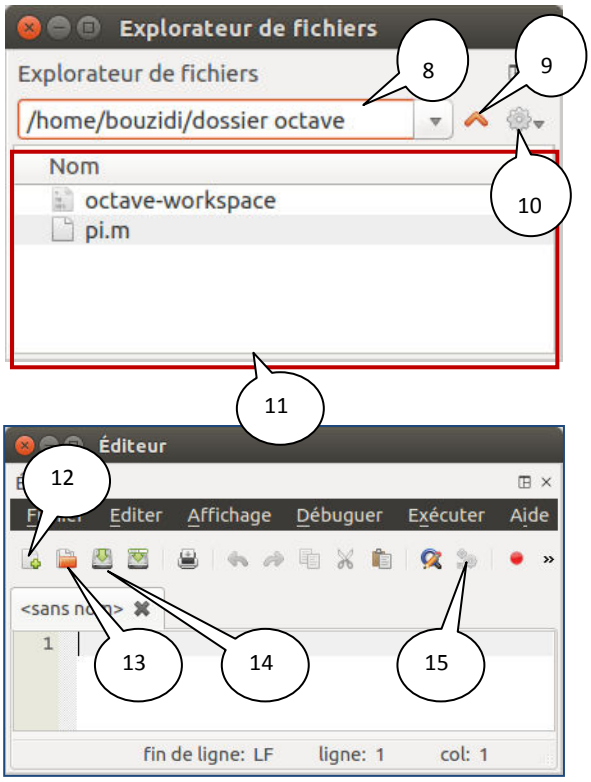
Remarquez, dans toutes les fenêtres précédentes, un certain nombre d'éléments visuels. Je vous invite à découvrir certains parmi eux en les manipulant :

2 – Identifier quelques éléments visuels

Indiquez à quoi servent les éléments visuels suivants :



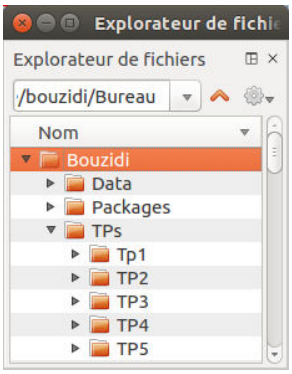




1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	
11.	
12.	
13.	
14.	
15.	
16.	

3 – Organisez votre travail

Lorsque vous vous engager à réaliser un projet d'analyse de données ou de résolution d'un problème scientifique, vous serez amenés à gérer des données, à écrire des commandes, à utiliser des packages et probablement à écrire des scripts. Afin de ne pas trop vous perdre, il est recommandé d'organiser votre dossier de travail. Ainsi, il faut que vous soyez capables de créer des répertoires, et de définir le répertoire courant. Pour faire cela, vous allez vous servir de l'explorateur de fichier. Vous pouvez aussi utiliser des commandes que vous taperez dans la fenêtre de commande. Je vous invite à créer une structure de dossier comme suit :



Indication : *Remplacez « Bouzidi » par votre nom.*

Répertoire courant : Placez-vous dans le répertoire TP1 afin de l'indiquez comme répertoire courant. Tous ce que vous allez faire durant cette séance doit se faire dans le répertoire TP1.

Disposition des fenêtres : Vous pouvez disposez à votre guise les fenêtres d'Octave.

- Disposition par défaut : Dans le menu fenêtre, cliquez sur « Rétablir la disposition par défaut des fenêtres ».
- Déplacez par glisser déposer les fenêtres comme il vous semble puis rétablir la disposition par défaut
- Faite afficher la fenêtre de commande en dehors d''Octave en utilisant l'icône suivante :
- Remettre la fenêtre de commande dans 'environnement Octave et utilisant l'icone suivante :

4 – Gérer l'historique des commandes

Effacement de l'historique : Effacer l'historique des commandes en tapant dans la fenêtre de commande : **history –c**

Tapez « **help history** » pour comprendre comment utiliser cette commande.

Questions :

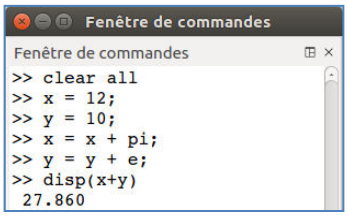
Indiquez la commande permettant d'afficher les 3 dernières commandes :.....
.....

Indiquez comment je vais faire pour sauver l'historique des commandes dans le fichier nommé « *CommandeTP1* » :.....
.....

Indiquez comment je vais faire pour lire l'historique des commandes depuis le fichier nommé « *CommandeTP1* » :.....
.....

5 - Gestion de l'espace de travail : L'espace de travail englobe vos variables. A chaque fois que vous créez des variables, celle-ci sont rajoutées dans cet espace. Vous pouvez supprimer des variables ou les renommer.

Saisissez les commandes suivantes



Questions :

Que fait « clear all » : .....
.....

Regardez dans l'espace de travail, quelles sont les variables que vous voyez ?
.....

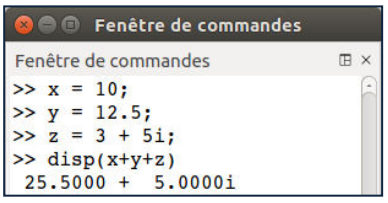
Tapez la commande « clear x » et regardez de nouveau l'espace de travail, que contient-il ?
.....

6 – Interagir à travers la fenêtre de commande :

C'est depuis cette fenêtre que vous commandez Octave. Vous pouvez lui demander ce que vous voulez : créer des variables, organiser vos répertoires, demander de l'aide, installer de nouveaux packages, etc. …

Nettoyage de la fenêtre de commandes : Vous pouvez nettoyer la fenêtre de commandes en tapant « **clc** ». Attention, cette commande ne supprime pas les commandes (qui restent d'ailleurs visibles dans la fenêtre d'historique) mais rend seulement l'affichage plus clair.

Création de nouvelles variables : Il suffit d'affecter des valeurs à des noms pour créer des variables. Je vous invite à créer 3 variables, x, y et z, respectivement initialisées à 10, 15.5 et 3+5i.



Effacez la fenêtre de commande à l'aide de la commande « **clc** ».

Reprenez les commandes de création des variables x, y et z mais sans mettre de « ; » à la fin. Que remarquez-vous : .....

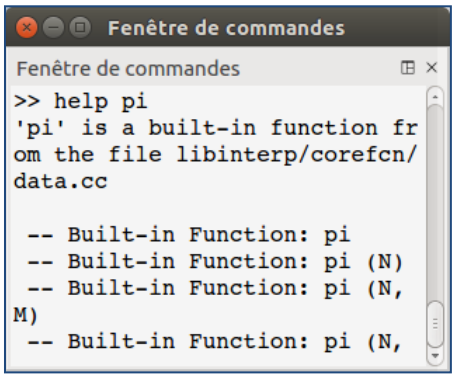
.....

.....

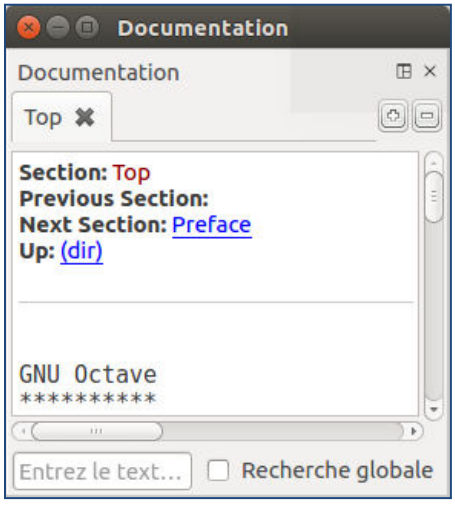
7 – Demandez de l'aide

Vous avez au moins 2 façons de demander de l'aide à Octave : Tapez la commande **help** ou **doc**.

« help » documente une fonction ou commande que vous indiquez. Par exemple si vous voulez savoir c'est quoi ***pi***, vous devez taper « ***help pi*** » qui explique c'est quoi « ***pi*** » :



« doc », vous donne accès à une documentation détaillée sur Octave. Tapez cette commande pour voir :



8 – Gérer les packages

Les packages sont des bibliothèques riches vous permettant d'étendre les fonctionnalités d'Octave. Ainsi, par exemple, vous pouvez avoir accès à des fonctions de traitement statistique avec le package « *statistics* » ou de traitement symbolique avec le package « *symbolic* ».

Se procurer un package :

C'est dans le site web officiel d'Octave (http://octave.sourceforge.net/) que vous pouvez vous procurer des packages. Cependant, afin de vous éviter de vous connecter sur Internet, dans votre cours en ligne, j'ai déjà déposé plusieurs packages d'Octave. Je vous demande de télécharger le package de traitement statistique (statistics-1.3.0.tar.gz) et de le déposer votre répertoire courant (TP1).

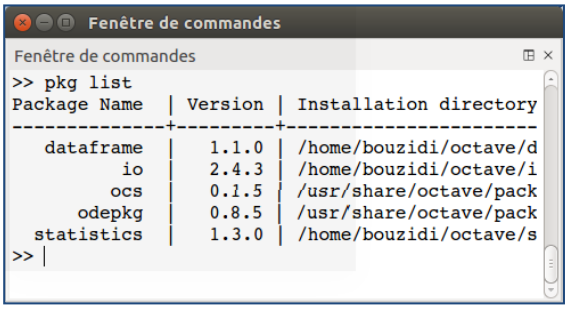
Installation d'un package :

Vous utiliserez la commande « pkg ». Ainsi pour installer le package sur les statistiques vous tapez la commande suivante :

pkg install statistics-1.3.0.tar.gz

Lister les packages installés :

Tapez « pkg list »



Désinstallation d'un package :

Tapez la commande « help pkg » pour comprendre comment utiliser la commande « pkg ». A vous de vous débrouiller pour désinstaller le package « statistics-1.3.0.tar.gz ». Notez le nom du package dans la liste qu'Octave vous a affiché.

Donnez la commande que vous allez utiliser :

.....

.....

Chapitre 2 –Généralités

A la fin de ce chapitre, vous serez capables :

- de Faire des calculs sur les nombres
- de manipuler des variables et des fonctions
- d'expliquer comment Octave représente-t-il les nombres
- de manipuler des séquences de nombres
- de manipuler quelques valeurs particulières
- de manipuler des chaines de caractères

II.1 – Types de nombres

On distingue trois types de nombres : les réels, les entiers et les complexes.

☞ **Les réels** :

On distingue deux représentations : « double » et « simple ». De façon interne, Octave stocke par défaut tous les nombres en virgule flottante "**double précision**" (au format IEEE qui occupe 8 octets par nombre, donc 64 bits). Les nombres ont une précision finie de 16 chiffres décimaux significatifs, et une étendue allant de 10^{-308} à 10^{+308}.

Les nombres réels seront saisis par l'utilisateur selon les conventions de notation décimale standard (si nécessaire en notation scientifique). Voici des exemples de nombres réels valides :

5, -39, 0.000365, -1.6341e11, 4.531e-15

Il est toutefois possible de réduire explicitement la précision des nombres en utilisant la fonction « **single**() ». Notez bien que ceci est fait, en général pour gagner de l'espace mémoire tout en perdant en précision. En effet, les nombres en « simple précision » sont codés sur 32 bits alors que les doubles précisions le sont sur 64 bits. En simple précision les nombres ont 7 chiffres décimaux significatifs, et une étendue allant de 10^{-38} à 10^{+38}.

Il est possible de convertir un nombre représenté en simple précision vers la double précision en utilisant la fonction « double() »

Attention !

Lorsque vous combinez les deux représentations dans une expression, le résultat sera toujours donné dans la représentation la moins précise (ici : simple précision !).

☞ **Les entiers** :

On vient de voir que Octave manipule, par défaut, les nombres sous forme réelle en virgule flottante (double précision ou, sur demande, simple précision). Ainsi l'expression « *n = 432* » stocke de façon interne le nombre *n* spécifié sous forme de variable réelle double précision, bien que l'on ait saisi un nombre entier.

On peut, cependant, manipuler des variables de types entiers représentés sur 8, 16, 32 et 64 bits. Pour cela on doit nous servir des fonctions de conversions suivantes :

- Les fonctions de conversion int8, int16, int32 et int64 génèrent des variables entières signées stockées respectivement sur 8 bits, 16 bits, 32 bits ou 64 bits ;
- les fonctions de conversion uint8, uint16, uint32 et uint64 génèrent des variables entières non signées (*unsigned*).

Remarque :

- Les opérations arithmétiques sur des entiers sont plus rapides que les opérations analogues réelles.
- Les fonctions de conversion des réels en entier arrondissent au nombre le plus proche (équivalent de la fonction round() que nous verrons ultérieurement)

Attention :

Lorsque vous utilisez des opérateurs ou des fonctions mélangeant des opérandes de types entiers et réels, le résultat retourné sera toujours de type entier ! Si vous ne souhaitez pas ça, vous devrez convertir au préalable l'opérande « *entier* » en « *réel* » double précision (avec ***double(entier)***) ou simple précision (avec ***single(entier)***) !

Il faut être vigilant lors des conversions. N'oubliez pas que lorsque vous convertissez un nombre réel vers une représentation entière vous réduisez la précision et possiblement, le nombre que vous souhaitez représenter n'est tout simplement non représentable dans le codage que vous souhaitez. Par exemple int8(300) vous rendra la valeur +127. Octave essaye de convertir le réel 300.0 vers une représentation entière (signée) sur 8 bits. Dans cette représentation la plage des valeurs possibles est de -128 à +127. Comme la valeur entière sur 8 bits la plus proche de 300.0 est +127, le résultat de cette conversion vous donne donc +127 !

☞ **Les nombres complexes** :

Octave manipule des nombres complexes. Ces nombres sont stockés de façon interne sous forme de réels « double précision » sur 2x 8 octets. Les 8 premiers octets pour la partie réelle et les seconds pour la partie imaginaire.

Voici quelques exemples d'écriture valides de nombres complexes :

4e-13 - 5.6i	-45 + 5*j	3 + i
3 - j	(14.5+5) + 13j	13+14 + 3i

Octave vous offre quelques fonctions très utiles pour manipuler des nombres complexes :

Fonction	Description
real(nb_complexe) imag(nb_complexe)	Retourne la partie réelle du nb_complexe spécifié, respectivement sa partie imaginaire Exemple : real(14+5i) retourne 14 imag(3+4i) retourne 4
conj(nb_complexe)	Retourne le conjugué du nb_complexe spécifié Exemple : conj(5+3i) retourne 5-3i
abs(nb_complexe)	Retourne le module du nb_complexe spécifié Exemple : abs(3+4i) retourne 5
arg(nb_complexe)	Retourne l'argument du nb_complexe spécifié Exemple : arg(12-4i) retourne -0.32175
isreal(var), iscomplex(var)	Permet de tester si l'argument (scalaire, tableau) contient des nombres réels ou complexes

II.2 – Variables et expressions

Variables : Les variables crées lors d'une session de travail Octave (interactivement ou depuis un script) résident en mémoire dans ce qu'on appel le « ***workspace*** ». Nul besoin de déclarer une variable, Octave la crée dès qu'il rencontre son nom dans la partie gauche d'une affectation. Il déduit son type et l'espace à lui allouer dans le *workspace* lors de l'évaluation de l'expression se trouvant dans la partie droite d'une affectation.


Attention :

Le nommage des variables doit respecter des règles précises :

- Un nom valide consiste en une lettre suivie de lettres, chiffres ou caractères souligné "_".
- Les lettres doivent être dans l'intervalle a-z et A-Z
- Les caractères accentués ne sont pas autorisés
- Le nombre maximum de caractères dans un nom est 63
- Les noms sont sensibles à la casse (prise en compte des majuscules et minuscules)

Remarque : la fonction ***namelengthmax*** renvoi le nombre maximum que pourrait avoir un nom d'une variable.

Nous avons vu, dans le chapitre précédent, que l'on peut gérer les variables par le biais que quelques fonctions : ***who***, ***clear*** et ***save*** par exemple. Ainsi, vous pourrez voir la liste de vos variables (de votre *workspace*) avec la fonction « *who* », en supprimer avec « *clear* » et en sauver sur disque avec la commande « *save* ».

Remarque : Vous pouvez utiliser des caractères de substitution « ***** » (Remplace 0, 1 ou plusieurs caractères quelconques) et « **?** » (Remplace 1 caractère quelconque) pour désigner un sous groupe de variables (créer un filtre en quelques sortes). Par exemple la commande « **who ?x** » affiche toutes les variables composées de 2 lettres dont la seconde est « x ». Par contre, la commande « **who *x** » affiche toutes les variables dont les noms se terminent par « x ».

Expressions : Une "expression" Octave est une construction valide faisant usage de nombres, de variables, d'opérateurs et de fonctions.

Exemples : pi*r^2 , sqrt((b^2)-(4*a*c))

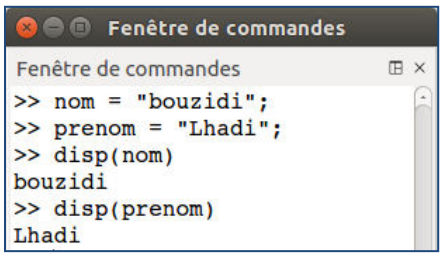
Commandes de gestion des variables :

- affectation avec affichage du résultat.
  Exemple : « X = 12+sin(30) »
- affectation sans affichage du résultat.
  Exemple : « X = 12+sin(30) ; »
  Notez bien que c'est le point virgule qui précise à Octave de ne pas afficher le résultat !
- Expression : Si l'on écrit une expression sans indiquer de nom de variable réceptrice, Octave crée une variable par défaut qu'il nomme « **ans** » (*ans* pour *answer*).
- Affichage d'une variable en tapant son nom
- « who » et « whos » permettent de lister des variables
- « clear » permet de supprimer des variables

II.3 Chaines

Il est tout à fait possible de manipuler des chaines de caractères dans Octave. Celles-ci doivent être délimitées par deux apostrophes ou des guillemets.

Exemple :



Octave représente une chaine sous la forme d'un vecteur de caractères. Ainsi, dans l'exemple ci-dessus, *nom(1)* me renvoi le caractère « *b* ».

Remarque : On verra plus tard qu'Octave commence ses indexes à partir de 1 à la différence de plusieurs langages comme *C* ou *Python* qui commencent de 0.

Si une chaine doit contenir des apostrophes ou des guillemets, il va falloir les dédoubler. Par exemple dans mon prénom « L'hadi », il y a une apostrophe, je dois donc écrire :

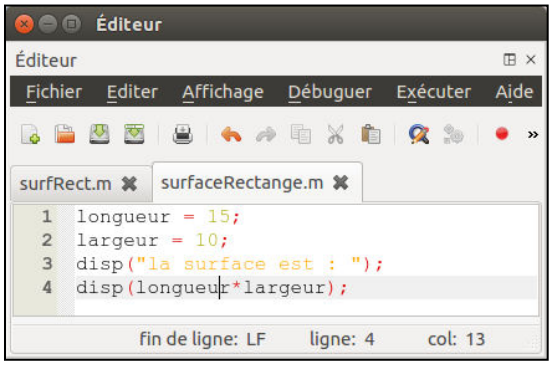
```
>> prenom = 'L''hadi'
prenom = L'hadi
```

II.4 Scripts

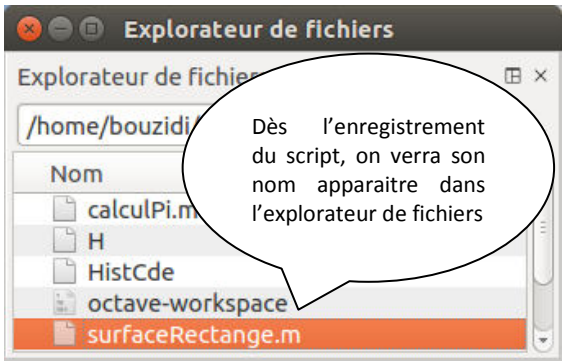
Ce sont des séquences de commandes sauvées sur disque sous forme de fichiers ayant comme extension « ***.m*** ». Ces fichiers sont appelés « ***m-files*** ». Pour les réutiliser, il suffit d'indiquer le nom du fichier « *m-file* » à partir d'une fenêtre de commandes ou à partir d'une fonction ou d'un autre script.

Attention : Il faut que le script soit accessible, c'est-à-dire, il doit se trouver dans le répertorie courant ou bien il faut indiquer le chemin complet au fichier de ce script.

Exemple : avec l'éditeur d'Octave (ou n'importe quel autre éditeur de textes), on peut saisir un script permettant de calculer la surface d'un rectangle :



Ce script est sauvé dans le répertoire courant sous le nom de « **sufaceRect.m** ».



Pour exécuter ce script, il suffit de taper son nom dans la fenêtre de commande. Vous pouvez aussi faire la même chose à partir de l'explorateur de fichiers en faisant un clic-droit de la souris sur le nom du script puis en choisissant dans le menu contextuel « «ouvrir » :

II.5 Fonctions

Ce sont les fonctions qui font la richesse et la puissance d'Octave. Comme c'est un langage crée pour le calcul scientifique, il fourni un grand nombre de fonctions mathématiques prédéfinis, mais l'utilisateur peut en créer aussi. Les fonctions respectent les mêmes règles de nommages que les variables.

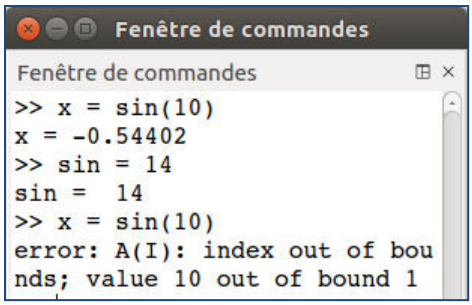
Dans ce qui suit, nous présenterons les fonctions prédéfinies, mais les fonctions crées par les utilisateurs seront abordé dans le chapitre 8 (écrire des programmes interactifs).

Fonctions prédéfinies (*builtin functions*)

Elles existent déjà dans le noyau d'Octave. Vous n'avez pas besoin, ni de les définir, ni de les importer ou d'installer de nouveau packages.

Les noms des fonctions prédéfinies (*builtin functions*) sont en minuscules. Par exemple « ***cos()*** » est la fonction rendant le cosinus d'un angle, mais « ***COS()*** » n'est pas une fonction prédéfinie !

Attention : Les noms de fonctions ne sont pas réservés ! Il est tout à fait possible d'avoir cette situation :



Dans l'exemple ci-dessus, on voit dans la première commande « **x= sin(10)** » fonctionnant correctement car on fait appelle à la fonction *sinus*. Mais en faisant « **sin=14** », on a utilisé (dans notre espace de travail) le nom « *sin* » pour identifier une variable contenant la valeur 14. Ainsi, lorsqu'on a demandé à Octave d'exécuter la dernière commande « **x= sin(10)** », il nous renvoi une erreur car « **sin** » pour lui n'est plus le nom d'une fonction *builtin*, mais d'une variable que l'utilisateur a crée. Pour rétablir le nom de la fonction sinus, il faut supprimer la variable « sin » en écrivant « **clear sin** ».

Octave nous fournie une large liste de fonctions prédéfinies :

Fonction	Description
sqrt(*var*)	Racine carrée de *var*. Remarque : pour la racine *n*-ème de *var*, faire *var*^(1/*n*)
exp(*var*)	Exponentielle de *var*
log(*var*) log10(*var*) log2(*var*)	Logarithme naturel de *var* (de base **e**), respectivement de base **10**, et de base **2** Ex: log(exp(1)) => 1, log10(1000) => 3, log2(8) => 3
cos(*var*) acos(*var*)	Cosinus, resp. arc cosinus, de *var*. Angle exprimé en radian
sin(*var*) asin(*var*)	Sinus, resp. arc sinus, de *var*. Angle exprimé en radian
sec(*var*) et csc(*var*)	Sécante, resp. cosécante, de *var*. Angle exprimé en radian
tan(*var*) et atan(*var*)	Tangente, resp. arc tangente, de *var*. Angle exprimé en radian
cot(*var*) et acot(*var*)	Cotangente, resp. arc cotangente, de *var*. Angle exprimé en radian
atan2(*dy*,*dx*)	Angle entre -pi et +pi correspondant à *dx* et *dy*
cart2pol(*x*,*y* {,*z*}) et pol2cart(*th*,*r* {,*z*})	Passage de coordonnées cartésiennes en coordonnées polaires, et vice-versa
cosh, acosh, sinh, asinh, sech, asch, tanh, atanh, coth, acoth	Fonctions hyperboliques...
factorial(*n*)	Factorielle de *n*
rand rand(*n*) rand(*n*,*m*)	Génération de nombres aléatoires **réels** compris entre **0.0 et 1.0** selon une distribution uniforme standard : - génère un nombre aléatoire - génère une matrice carrée *n* x *n* de nombres aléatoires - génère une matrice *n* x *m* de nombres aléatoires
fix(*var*) round(*var*) floor(*var*) ceil(*var*)	Troncature à l'entier, dans la direction de zéro (donc 4 pour 4.7, et -4 pour -4.7) Arrondi à l'entier le plus proche de *var* Le plus grand entier qui est inférieur ou égal à *var* Le plus petit entier plus grand ou égal à *var* Ex: fix(3.7) et fix(3.3) => 3, fix(-3.7) et fix(-3.3) => -3 round(3.7) => 4, round(3.3) => 3, round(-3.7) => -4, round(-3.3) => -3 floor(3.7) et floor(3.3) => 3, floor(-3.7) et floor(-3.3) => -4 ceil(3.7) et ceil(3.3) => 4, ceil(-3.7) et ceil(-3.3) => -3
mod(var1,var2) rem(var1,var2)	Fonction *var1* "modulo" *var2* Reste ("remainder") de la division de *var1* par *var2* Remarques: - *var1* et *var2* doivent être des scalaires réels ou des tableaux réels de même dimension - rem a le même signe que *var1*, alors que mod a le même signe que *var2* - les 2 fonctions retournent le même résultat si *var1* et *var2* ont le même signe Ex: mod(3.7, 1) et rem(3.7, 1) retournent 0.7, mais mod(-3.7, 1) retourne 0.3, et rem(-3.7, 1) retourne -0.7
idivide(*var1*, *var2*, '*regle*')	Division entière. Fonction permettant de définir soi-même la *règle* d'arrondi.
abs(*var*)	Valeur absolue (positive) de *var* Ex: abs([3.1 -2.4]) retourne [3.1 2.4]
sign(*var*)	(signe) Retourne "1" si *var*>0, "0" si *var*=0 et "-1" si *var*<0 Ex: sign([3.1 -2.4 0]) retourne [1 -1 0]
real(*var*) et imag(*var*)	Partie réelle, resp. imaginaire, de la *var* complexe

QCM 2 – Généralités

Q1 : Octave prendre en charge les types de nombres suivants :

- ☐ Réels
- ☐ Complexes
- ☐ Entiers

Q2 : Pour chaque type de nombre, *Octave* permet plusieurs représentations. Par exemple, pour les réels on peut avoir la simple précision et la double précision :

☐ Vrai ☐ Faux

Q3 : La représentation en double précision (**double**) suppose que les nombres sont:

- ☐ Des entiers représentés sur 32 bits
- ☐ Des entiers représentés sur 64 bits
- ☐ des réels représentés sur 32 bits en virgule flottante
- ☐ des réels représentés sur 32 bits en virgule fixe
- ☐ des réels représentés sur 64 bits en virgule flottante
- ☐ des réels représentés sur 32 bits en virgule flottante

Q4 : Relier les types de représentation aux nombres :

double	Entiers sur 8 bits
simple	Entiers sur 16 bits
int8	Entiers sur 32 bits
int16	Entiers sur 64 bits
int32	Entiers non signé sur 8 bits
unt64	Entiers non signé sur 16 bits
uint8	Entiers non signé sur 32 bits
uint16	Entiers non signé sur 64 bits
uint32	Réel en virgule flottante sur 32 bits
uunt64	Réel en virgule flottante sur 64 bits

Q5 : Indiquez les notations correctes pour les nombres suivants:

- ☐ 12,5
- ☐ 14.5
- ☐ 13+i5
- ☐ 13+5i
- ☐ 13+5*i

Q6 : Le nombre (**12 + int32(14)**) est un :

- ☐ double
- ☐ simple
- ☐ int8
- ☐ int16
- ☐ int32
- ☐ int64
- ☐ uint8
- ☐ unint16
- ☐ uint32
- ☐ uint64
- ☐ complex

Q7 : Le nombre (**12 + single(14)**) est un nombre en :

- ☐ double
- ☐ simple
- ☐ int8
- ☐ int16
- ☐ int32
- ☐ int64
- ☐ uint8
- ☐ unint16
- ☐ uint32
- ☐ uint64
- ☐ complex

Q8 : Le nombre (**12 + complex(14)**) est un nombre en :

- ☐ double précision
- ☐ simple précision
- ☐ int8
- ☐ int16
- ☐ int32
- ☐ int64
- ☐ uint8
- ☐ unint16
- ☐ uint32
- ☐ uint64
- ☐ complex

Q9 : La fonction « **namelengthmax** » retourne :

- ☐ le nombre maximum de variables que vous pouvez définir dans un script
- ☐ Le nombre maximum de caractères que vous pouvez saisir dans un script
- ☐ La longueur maximale que peut avoir un identificateur (nom de variable, de fonction, etc.)

Q10 : Lorsque je saisie, dans la fenêtre de commande, l'instruction suivante : « **a = 12** ». La variable « a » sera représentée en simple précision (32 bits) :

□ Vrai □ Faux

Q11 : Lorsque je saisie, dans la fenêtre de commande, l'instruction suivante : « **a = int8(12)+3i** ». La variable « a » sera représentée sur 16 octets (2 doubles) :

□ Vrai □ Faux

Q12 : Indiquez combien vaut la variable « ***a »*** à l'issue de la commande suivante : « **a = int8(400)** » : .....

Q13 : Indiquez combien vaut la variable « ***a »*** à l'issue de la commande suivante : « **a = uint8(-15)** » : .....

Q14 : Indiquez combien vaut la variable « ***a »*** à l'issue de la commande suivante : « **a = uint8(105.75)** » : .....

Q15 : En ne donnant que 2 chiffres après la virgule, Indiquez combien vaut la variable « ***a »*** à l'issue de la commande suivante : « **a = pi + 2** » : .....

Q16 : Indiquez combien vaut la variable « ***a »*** à l'issue des commandes suivantes : .....

```
>> pi = 10;
>> a = pi+2;
```

Q17 : Indiquez combien vaut la variable « ***a »*** à l'issue des commandes suivantes : .....

```
>> pi = 10;
>> clear pi
>> a = pi+2;
```

Q18 : Donnez la commande permettant d'afficher la liste des variables de l'espace de travail dont le nom commence pas « ***e*** » : .....

Q19 : Donnez la commande permettant d'afficher la liste des variables de l'espace de travail dont les noms ont comme second caractère « ***t*** » : .....

Q20 : Donnez la commande permettant de supprimer toutes les variables de l'espace de travail dont le nom se termine par le caractère « ***e*** » : .....

Q21 : Le fait de rajouter un « ; » à la fin d'une commande indique à Octave qu'il ne faut pas afficher le résultat de cette commande immédiatement :

□ Vrai □ Faux

Q22 : Je suppose que j'ai crée une variable « promo » comme suit : « **promo = "Analyse proba"**». Que va m'indique la commande suivante : « **typeinfo(promo)**»

- □ chaine
- □ string
- □ caractères
- □ double
- □ array

Q23 : Je suppose que j'ai crée une variable « promo » comme suit : « **promo = "Analyse proba"**». Que va m'indique la commande suivante : « **promo(2)**»

- □ A
- □ An
- □ Analyse
- □ proba
- □ n

Q24 : Je suppose que j'ai crée une variable « **promo** » comme suit : « **promo = "Analyse proba"**». Que va m'indiquer la commande suivante : « **promo(1 :7)**»

- □ A
- □ An
- □ Analyse
- □ proba
- □ n

Q25 : Un script Octave est

- □ une séquence de commandes sauvée sur disque sous l'extension « .oct »
- □ une séquence de commandes sauvée sur disque sous l'extension « .m »
- □ un programme binaire

Q26 : Une fonction est un script particulier dans lequel on définit le nom de la fonction, des arguments d'entrée et des résultats en sortie et un ensemble de commandes, le tout sauvé sur disque sous le nom de la fonction suivi de l'extension « .m ».

□ Vrai □ Faux

Q27 : Une fonction prédéfinie (*builtin*) est un script que l'on peut appeler à tout moment. Elle renvoi, en général un résultat et peut exiger des arguments. Son nom est en minuscules. « sin », « cos », « pi » en sont des exemples.

□ Vrai □ Faux

TP2 – Manipuler des nombres

Octave prend en charge plusieurs types de nombres : réel (double et simple précision), entiers (allant de 1 octet à 8 octets) et complexes. Je vous invite à les découvrir.

Exo 1 : Affichage des nombres

Tapez les commandes suivantes :

```
Fenêtre de commandes
>> x = 70000;
>> y = 7e4;
>> z = 0.07e6;
>> t = 700000e-1;
```

Q1 : Que constatez-vous :

- ☐ Les variables sont toute des entiers
- ☐ Les variables contiennent toutes la même valeur
- ☐ ***x*** est en notation scientifique
- ☐ ***y*** est en notation scientifique
- ☐ le « **;** » permet de ne pas afficher le résultat directement
- ☐ les variables « ***y***, ***z*** et ***t*** » sont des complexes
- ☐ les variables sont représentées en interne en virgule flottante sur 32 bits (simple précision)

Je vous invite maintenant à effacer la fenêtre de commande en tapant la commande « ***clc*** ». Tapez les commandes suivantes :

```
Fenêtre de commandes
>> x = 123456789123456789.123456789;
>> format short
>> disp(x)
   1.2346e+17
>> format long
>> disp(x)
 123456789123456784
>> format short e
>> disp(x)
   1.2346e+17
>> format long e
>> disp(x)
   1.23456789123457e+17
```

Dans les manipulations ci-dessus, vous utilisez une commande appelée « **format** », elle permet de spécifier le format d'affichage des nombres (attention, elle ne modifie pas la représentation interne de vos nombres !, c'est juste au niveau de l'affichage). En utilisant la commande « **help format** » répondez aux questions suivantes :

Q2 : « format short » permet d'afficher les nombres :

- ☐ sur au plus 10 caractères avec 5 chiffres significatifs
- ☐ sur au plus 20 caractères avec 15 chiffres significatifs

Q3 : « format long » permet d'afficher les nombres :

- ☐ sur au plus 10 caractères avec 5 chiffres significatifs
- ☐ sur au plus 20 caractères avec 15 chiffres significatifs

Q4 : Octave utiliser la notation scientifique dès qu'il n'arrive pas à afficher les nombres en notation décimale normale :

☐ Vrai ☐ Faux

Exo 2 : Nombres réels

Octave utilise, en interne 2 représentations : « double » et « simple ». « double » est la représentation par défaut.

Q5 : Dans la représentation « double précision», Octave utilise :

- ☐ 16 bits
- ☐ 32 bits
- ☐ 64 bits

Q6 : Dans la représentation « simple précision », Octave utilise :

- ☐ 16 bits
- ☐ 32 bits
- ☐ 64 bits

Q7 : A l'issue des commandes suivantes :

```
Fenêtre de commandes
>> x = 14;
>> y = single(x^2);
```

- ☐ x est en double précision
- ☐ x est en simple précision
- ☐ x occupe 64 bits
- ☐ x occupe 32 bits
- ☐ x occupe 8 octets
- ☐ x occupe 4 octets

Q8 : A l'issue des commandes suivantes :

```
Fenêtre de commandes
>> x = 14;
>> y = single(x^2);
```

- y est en double précision
- y est en simple précision
- y occupe 64 bits
- y occupe 32 bits
- y occupe 8 octets
- y occupe 4 octets

Q9 : A l'issue des commandes suivantes, l'affichage de ***y*** donne une valeur bizarre : « **Inf** », qu'est ce que cela veut dire ?

```
Fenêtre de commandes
>> x = 14e128;
>> y = single(x);
>> disp(y)
Inf
```

- ***x*** contient une valeur plus grande que la plus grande valeur représentable en simple précision. Octave considère que ***single(x)*** est infinie
- Octave a fait une erreur. En principe, il doit afficher correctement la valeur de y !
- ***« Inf »*** veut dit « inférieur », c'est-à-dire que la valeur de y est inférieur à celle de x

Q10 : A l'issue des commandes suivantes, l'affichage de ***x*** donne une valeur bizarre : « **NaN** », qu'est ce que cela veut dire ?

```
Fenêtre de commandes
>> x = 0/0;
warning: division by zero
>> disp(x)
NaN
```

- ***x*** contient une chaine de caractères
- Octave a fait une erreur. En principe, il doit afficher correctement la valeur de x !
- ***x*** ne doit pas exister car il est issu d'une erreur de calcul
- ***NaN*** *veut not a number.* Cela veut dire qu'Octave produit un résultat numérique erroné

Exo3 - Les entiers

Afin d'optimiser le stockage et le traitement des données, Octave permet de convertir les réels (virgule flottante) en entier sur 8, 16, 32 et 64 bits et cela avec ou sans signe.

Essayez les commandes suivantes : ⟹

```
Fenêtre de commandes
>> x = 350;
>> a = int8(x);
>> b = int16(x);
>> c = int32(x);
>> d = int64(x);
>> disp(a)
127
>> disp(b)
350
>> disp(c)
350
>> disp(d)
350
```

Q11 : On voit que la variable « **a** » ne contient pas exactement la valeur qu'elle devrait contenir (350). Cela est du au

- ***Int8*** converti le nombre 350 en entier sur 8 bits en représentation en complément à 2. On sait que cette représentation, sur 8 bits, elle permet de représenter les valeurs incluses dans l'intervalle [-128, +127]. Comme 350 est plus grande que +127, Octave nous donne cette valeur
- C'est tout simplement un bug
- La fonction ***int8*** fait n'importe quoi !, ce n'est pas logique 350 n'est pas 127 !

Q12 : Saisissez les commandes suivantes : ⟹

```
Fenêtre de commandes
>> x = 350;
>> y = int8(10)+ x
y = 127
```

On voit que la variable « **y** » ne contient pas exactement la valeur qu'elle devrait contenir (10+350). Cela est du au

- C'est tout simplement un bug
- La fonction ***int8*** fait n'importe quoi !, ce n'est pas logique int8(10)+350 = 360 ce n'est pas 127 !
- X est un double, int8(10) est un entier sur 8 bits. Le mélange donne un int8. Comme la valeur qu'on essaye de représente est trop grande, Octave nous donne la plus grande valeur qu'il peut représenter sur 8bits : +127.

Exo4 - Les complexes

Les nombres complexes sont pris en charge par Octave qui offre quelques fonctions de manipulation de ces nombres.

Q13 : Saisissez les commandes suivantes :

```
Fenêtre de commandes
>> c1 = 12 + 3i;
>> c2 = 12 + 3*i;
>> c3 = 12 + 3j;
>> c4 = 12 + 3*j;
```

Les variables C1, c2, c3 et c4 :

- ☐ contiennent toute un nombre complexe
- ☐ sont toutes égales
- ☐ ont des types différents
- ☐ C3 et c4 sont fausse en principe elles doivent comporter « i » au lieu de « j »

Q14 : Saisissez les commandes suivantes :

```
Fenêtre de commandes
>> c = 12 + 3i;
>> a = real(c);
>> b = imag(c);
>> c = conj(c);
>> d = abs(c);
>> e = arg(c);
```

Indiquez à quoi correspond chacune des variables « a », « b », « c », « d » et « e » :

Variable	Ce quelle contient
a	
b	
c	
d	
e	

Indication : pour manipuler un nombre complexe, on peut avoir besoin de sa partir réelle, de sa partie imaginaire, de son conjugué, de son argument et de son module !

Q15 : Saisissez les commandes suivantes :

```
Fenêtre de commandes
>> x = complex(12,5);
>> y = complex(12);
>> z = complex(i);
>> t = complex(j);
```

Indiquez à quoi correspond chacun des variables « x », « y », « z », et « t » :

- ☐ Contiennent toute un nombre complexe
- ☐ « z » et « t » sont différentes
- ☐ « y » est un réel
- ☐ La partie imaginaire de « y » est égale à 0.

Q16 : Saisissez les commandes suivantes :

```
Fenêtre de commandes
>> x = 12;
>> y = 13 + 5i;
>> z = x+y;
>> t = x*y;
```

Les variables « z », et « t » contiennent:

- ☐ Des réels
- ☐ Des entiers
- ☐ Des complexes

Q17 : Saisissez les commandes suivantes :

```
Fenêtre de commandes
>> c = 12+5i;
>> iscomplex(c)
ans =  1
>> isreal(c)
ans = 0
```

La fonction « iscomplex » nous renvoi la valeur « 1 », pourquoi ?:

- ☐ Car la variable « c » contient un complexe (« 1 » veut dire « vrai »)
- ☐ Car la variable « c » ne contient pas un complexe (« 1 » veut dire « faux »)
- ☐ Car la variable « c » contient un réel
- ☐ Car la variable « c » contient un imaginaire

La fonction « isreal » nous renvoi la valeur « 0 », pourquoi ?:

- ☐ Car la variable « c » ne contient pas un réel
- ☐ Car la variable « c » contient un complex
- ☐ Car la variable « c » contient un réel

Exo5 - Les strings

Q18 : Saisissez la commande suivante :

>> matiere = "Langage Octave";

☞ Donnez la commande permettant mettre dans une variable « **x** » le second caractère de la variable « matiere » :

.....

☞ Donnez la commande permettant mettre dans une variable « **y** » les 7 premiers caractères de la variable « matiere »....

☞ Donnez la commande permettant mettre dans une variable « **z** » les 6 derniers caractères de la variable « matiere » : .....

Chapitre 3 – Scalaires, séries, vecteurs et matrices

A la fin de ce chapitre, vous serez capables :

- D'identifier les objets manipulés par Octave
- Définir et utiliser des scalaires
- Définir et utiliser des séries
- Définir et utiliser des vecteurs
- Définir et utiliser des matrices

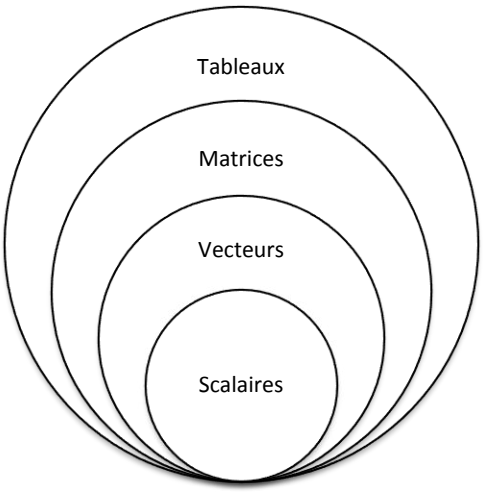
3.1 Divers types d'objets

Octave prend en charge plusieurs types d'objets hiérarchisés : scalaires, séries, vecteurs, chaines, matrices, tableaux multidimensionnels, structures et tableaux cellulaires.

Nous ne traiterons pas, dans ce qui suit, des tableaux multidimensionnels, des structures et des tableaux cellulaires.

Octave est fait à l'origine pour traiter des matrices et c'est la raison pour laquelle il se ramène à cette entité. Ainsi,

- Un scalaire est considéré comme une matrice particulière (une ligne et une colonne).
- Un vecteur « ligne » est une matrice ayant une seule ligne, mais plusieurs colonnes.
- Un vecteur « colonne » est une matrice ayant une seule colonne, mais plusieurs lignes.
- Une matrice est un tableau particulier (*array*) ayant plusieurs lignes et plusieurs colonnes.
- Un tableau à 3 dimensions peut être considéré comme plusieurs pages contenant chacune une matrice.
- On peut aussi imaginer des tableaux à plusieurs dimensions…



3.2 Scalaires

Octave ne différencie fondamentalement pas une matrice à N-dimension (tableau ou ***array*** en anglais) d'un vecteur ou d'un scalaire, et ses éléments peuvent être redimensionnés dynamiquement. Une variable scalaire n'est donc, en fait, qu'une variable matricielle de 1x1 élément (vous pouvez le vérifier avec la commande ***size(variable_scalaire)***).

Nous avons vu, dans le chapitre précédent, comment Octave manipule les nombres. En réalité ces nombres sont des scalaires.

Dans l'exemple, ci-dessous, j'ai crée un variable « ***x*** » initialisée avec la valeur 14. Rappelez-vous, on avait dit que, par défaut, la valeur des nombres est un « ***double*** ».

Mais c'est quoi exactement ce ***x*** ?

Fenêtre de commandes

```
>> x = 14;
>> disp(size(x))
   1   1
>> disp(isscalar(x))
 1
```

Voyons d'abord, la commande « ***size*** ». En voyant ce nom, vous allez vous attendre à obtenir une taille ou probablement le nombre d'octets occupés ou quelque chose comme ça. Mais en tapant « ***help size*** », vous serez surpris de découvrir qu'elle retourne le nombre de lignes et de colonnes d'une variable. On voit bien ici, qu'Octave se base sur les matrices ! En faisant « size(x) » octave a répondu (1 1) voulant dire : Une ligne et une colonne. En tapant « ***isscalar***(x) » octave a répondu « 1 » pour dire « OUI x est un scalaire !».

3.3 Séries (ranges)

Une série (ou range en anglais) est un moyen pratique d'écrire un vecteur-ligne composé de valeurs uniformément espacées. L'expression d'une série est définie par une valeur initiale, une valeur optionnelle définissant le pas entre deux valeurs successives et une valeur maximale. Ces trois valeurs sont séparées par l'opérateur « : » (*colon* en anglais). Si le pas n'est pas donné, il est supposé être égale à1.

Exemples :

- « 1 :5 » permet de définir un vecteur ligne composé des valeurs commençant par 1 et finissant par 5 avec un pas de 1, ce qui donne : [1,2,3,4,5]
- L'expression : « 1 :2 :5 » définit le vecteur composé des valeurs suivantes [1, 3, 5]

Notez bien qu'on peut avoir un pas négatif. Dans ce cas, la valeur spécifiée au début doit être supérieure à la valeur spécifiée à la fin. En effet, par exemple l'expression : « **10 :-2 :3** » définit les valeurs suivantes [10, 8, 6, 4].

Notez par ailleurs, que le « pas » peut être un réel. Par exemple l'expression : « **1 : 0.5 : 2** » définit les valeurs suivantes [1, 0.5, 1, 1.5, 2].

Remarque : Physiquement, Octave ne crée pas nécessairement un vecteur pour représenter une série. Il peut la régénérer par programme, ceci permet de réduire l'espace de stockage.

Lorsqu'on connait la valeur de départ et la valeur finale et l'on souhaite générer un nombre bien précis de valeurs entre ces deux bornes, Octave sait le faire à travers la fonction ***linspace***. Par exemple, si je veux générer 5 valeurs dans l'intervalle [0,1], j'utilise « linspace » comme ceci : « **linspace(0,1,5)** » :

```
Fenêtre de commandes
>> x = linspace(0,1,5);
>> disp(x)
   0.00000   0.25000   0.50000   0.75000   1.00000
```

On voit dans l'exemple ci-dessus que ***x*** contient une série de 5 valeurs comprises entre 0 et 1 !

Cette fonction est très utile pour générer les valeurs d'un axe d'un repère orthonormé par exemple.

Les séries générées par ***linspace*** sont linéaires. Si l'on souhaite générer des séries logarithmiques, Octave sait le faire à travers la fonction « **logspace** ».

Par exemple : x=logspace(2,6,5) crée x=[100 1000 10000 100000 1000000]

3.4 Vecteurs

Octave ne fait pas la différence entre un scalaire, un vecteur ou une matrice. Pour lui, ce sont tous des matrices plus ou moins particulières. Ainsi, un vecteur n'est rien d'autre qu'une matrice avec une seule ligne et plusieurs colonnes (1xM), ou une seule colonne et plusieurs lignes (Nx1). Dans le premier cas de figure (1xM), on parle de « vecteur ligne ». Dans le second cas de figure (Nx1), on parle de « vecteur colonne ».

Les éléments des vecteurs sont numérotés par des entiers débutant par la valeur 1 (et non pas 0, comme dans la plupart des autres langages de programmation).

Création d'un vecteur « ligne » : Il suffit d'indiquer des valeurs entre crochets et séparées par des espaces ou des virgules.

Par exemple si je veux créer le vecteur v1=(1,5,7,8)j'écrit la commande suivante :

```
>> v1 = [1 5 7 8]
```

Ou

```
>> v1 = [1 , 5 , 7 , 8]
```

Création d'un « vecteur-colonne » : Il suffit d'indiquer des valeurs entre crochets et séparées par des points-virgules. Ou la touche « ENTER ».

Par exemple si je veux créer le vecteur v2=(1,15,10,8), j'écrit la commande suivante :

```
>> v1 = [1 ; 15 ; 10 ;  8]
```

Ou

```
>> v1 = [1
         15
         10
          8]
```

Remarque : Il est tout à fait possible de construire un « vecteur ligne » à partir d'un « vecteur colonne » ou inversement en procédant par transposition. Octave a prévu l'opérateur « apostrophe » à cet effet. Ainsi, la transposé d'un vecteur « **v** » est « **v'** ».

```
Fenêtre de commandes
>> v1 = [1 ; 5; 7; 8]';
>> disp(v1)
   1   5   7   8
>> v2 = [1 15 10 8]';
>> disp(v2)
    1
   15
   10
    8
```

Comment accéder aux éléments d'un vecteur ?

C'est en spécifiant deux informations : le nom du vecteur suivi, entre parenthèses, d'indices :

vect(***indices***)

Que ce soit un « vecteur ligne » ou un « vecteur colonne », les indices permettent d'indiquer les valeurs auxquelles on voudrait accéder.

Les indices peuvent prendre plusieurs formes. Afin de les illustrer, nous allons prendre l'exemple d'un vecteur ligne composé des valeurs suivantes : v=[1,3,4,5,8,10,15]

Voici les différentes formes d'indices :

forme	exemple
Un seul indice qui indique une seul position dans le vecteur. v(i)	v(2) représente la valeur 3
séquence contiguë (série) d'indices allant de *ind1* jusqu'à *indN*. v(ind1 : indN)	v(3 : 5) représente les valeurs situées de la position 3 jusqu'à la position 5 : ce qui donne : 4 5 8
séquence d'indices de *ind1* à *indN* espacés par un pas. v(ind1 : pas : indN)	v(1 : 2 : 7) représente les valeurs situées de la position 1 jusqu'à la position 7 avec un pas de 2, ce qui donne les indices : [1 3 5 et 7] ce qui donne les valeurs: 1 4 8 15
Liste quelconque d'indices V([ind1, ind2, …, indN])	v([3 , 5 , 1]) représente les valeurs situées de la position 3, 5 et 1 ce qui donne les valeurs: 4 8 1

Remarque : pour représenter la borne supérieure de la série d'indices, on peut utiliser le mot « ***end*** » pour désigner le dernier élément du vecteur. Par exemple v(1 :2 :**end**) spécifie les éléments du vecteur ***v*** se trouvant aux positions 1, 3, 5, …, **end**. Avec ***end*** = *7*.

Initialiser un vecteur : On peut écrire une boucle (ok, je sais que je n'ai pas encore introduit les scripts et l'écriture des programme ! mais permettez-moi de faire une exception ici !) pour initialiser un vecteur.

Par exemple, si je veux initialiser à 0 les éléments du vecteur ***v*** se trouvant aux indices de 1 à 7 avec un pas de 2 (1, 3, 5 et 7), je procède comme suit :

```
for k=1:2:7
    v(k)=0
end;
```

Changer une valeur d'un vecteur : Il suffit d'utiliser l'affectation : « ***v(i) = valeur*** » avec ***i*** un indice. Par exemple si je veux mettre la valeur 15 dans la position 2 du vecteur, j'écris « **v(2)=15** ».

Changer simultanément plusieurs valeurs d'un vecteur : Il suffit d'utiliser l'affectation : « ***v(série) = valeur*** » avec « *série* » une série d' indices. Par exemple si je veux mettre la valeur 0 dans la position 1, 2 et 5 du vecteur ***v***, j'écris « **v([1,2,5])=0** ».

Comment supprimer totalement des valeurs d'un vecteur ?

C'est simple, c'est comme un changement simultané sauf que la valeur que l'on indique est un double crochet vide : « ***v(séquence) =[]*** ». Par exemple si je veux supprimer les valeurs se trouvant dans les positions « 1 » et « 3 » du vecteur v, j'écris : « **v([1, 3]) = []** ».

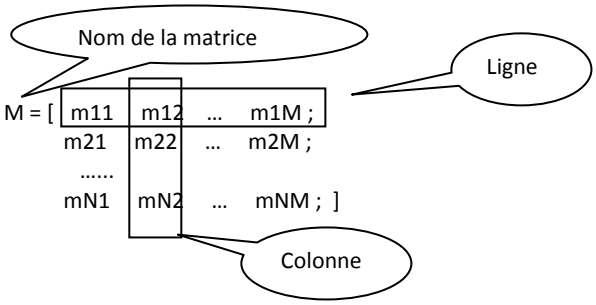
Connaitre la taille d'un vecteur : On utilisera la fonction « ***lenght*** ». Par exemple « length(v) », renvoi le nombre d'éléments de v.

3.5 Matrices

Une matrice est un tableau rectangulaire à 2 dimensions de *NxM* éléments (*N* lignes et *M* colonnes) de types nombres entiers, réels ou complexes ou de caractères.

Comme pour les vecteurs, la création d'une matrice se fait par une affectation d'éléments mis entre crochets. Les éléments des lignes sont séparés par un point-virgule ou la touche « *Enter* » ceux des colonnes sont séparés par des espaces, des virgules ou des tabulations exactement comme pour les vecteurs. Une matrice est donc une généralisation d'un vecteur.

Syntaxe :



Exemples : Matrice de 2 lignes fois 3 colonnes ayant pour valeurs : [-2 -1 0 ; 4 3 3]

Mathématiquement on peut l'écrire comme ceci :

$$M1 = \begin{pmatrix} -2 & -1 & 0 \\ 4 & 3 & 3 \end{pmatrix}$$

Voici une première façon de créer la matrice ***M1*** en utilisant l'espace comme séparateur de colonnes et le point-virgule comme séparateur de lignes :

```
Fenêtre de commandes
>> m1 = [-2 -1 0; 4 3 3]
m1 =

  -2  -1   0
   4   3   3
```

Voici une seconde façon de créer la matrice ***M1*** en utilisant des virgules comme séparateur de colonnes et le point-virgule comme séparateur de lignes :

```
Fenêtre de commandes
>> m1 = [-2 , -1 , 0; 4 , 3 , 3]
m1 =

  -2  -1   0
   4   3   3
```

Voici une troisième façon de créer la matrice ***M1*** en utilisant des espaces comme séparateur de colonnes et la touche « Enter » comme séparateur de lignes :

```
Fenêtre de commandes
>> m1 = [-2  -1  0
         4   3  3]
m1 =

  -2  -1   0
   4   3   3
```

On peut très bien créer la matrice ***M1*** en nous servant des séquences et de fonctions mathématiques comme suit :

```
Fenêtre de commandes
>> m1=[-2:0 ; 4 sqrt(9) 3]
m1 =

  -2  -1   0
   4   3   3
```

Créer des matrices par concaténation de vecteurs :

Il est très pratique de créer des matrices à partir de « vecteurs lignes » et de « vecteurs colonnes ».

Par exemple si je veux créer une matrice à partir de plusieurs vecteurs « colonnes », je fais comme suit :

Matrice = [Vcol1 Vcol2 … VcolM]

Avec « *Matrice* » le nom de ma matrice et « $Vcol_i$ » des vecteurs colonnes.

Si je veux créer une matrice à partir de plusieurs « vecteurs linge », je fais comme suit :

Matrice = [Vlig1 ; Vlig2 … VligN]

Avec « *Matrice* » le nom de ma matrice et « $Vlig_i$ » des « vecteurs lignes ».

Exemple 1 : Créer une matrice M à partir de 2 « vecteurs colonnes » :

```
Fenêtre de commandes
>> vcol1 = [1 ; 2 ; 3];
>> vcol2 = [5 ; 6 ; 7];
>> M = [vcol1, vcol2]
M =

    1    5
    2    6
    3    7
```

vcol1 vcol2

Exemple 2 : Créer une matrice M1 à partir de 2 « vecteurs lignes » :

```
Fenêtre de commandes
>> vlig1 = [1 2 3 4 5];
>> vlig2 = [7 8 9 10 11];
>> M1 = [vlig1;vlig2]
M1 =

     1     2     3     4     5
     7     8     9    10    11
```

vlig1 Vlig2

Attention aux incompatibilités !

La création de matrices à partir de « vecteurs lignes » ou de « vecteurs colonnes » exige que ces vecteurs aient la même taille sinon une erreur sera signalée :

```
Fenêtre de commandes
>> vlig1 = [1 2 3];
>> vlig2 = [7 8 9 10 11 14];
>> M1 = [vlig1;vlig2]
error: vertical dimensions mismatch (1x3 vs 1x6)
```

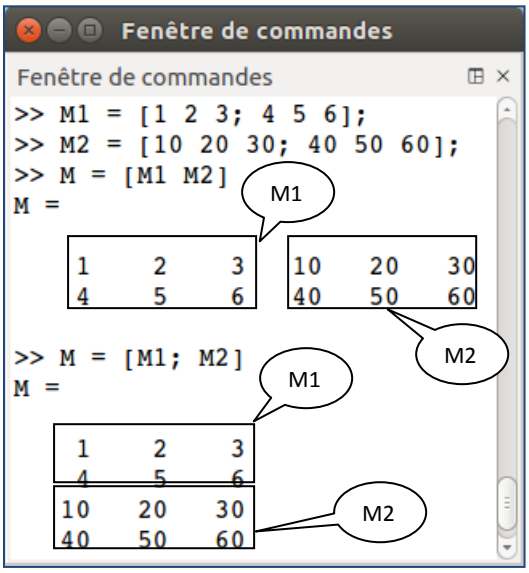
Création d'une matrice par concaténation de matrices :

On peut concaténer des matrices horizontalement (mises cote à cote) ou verticalement (mises les unes au dessous des autres).

Syntaxe : Horizontalement : M = [M1 M2 … Mn]

Syntaxe : Verticalement : M = [M1 **;** M2 **;** … Mn]

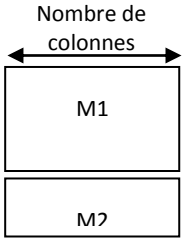
Exemple :



Attention aux incompatibilités !

La création de matrice par concaténation de matrices exige de la vigilance en ce qui concerne les dimensions de vos matrices d'origine :

> Si vous concaténer des matrices horizontalement, il faut que le nombre de lignes de ces matrices soit le même.
> Si vous concaténer des matrices verticalement, il faut que le nombre de colonnes de ces matrices soit le même.



Bien d'autres moyens de créer des matrices :

Il existe plusieurs fonctions ***Octave*** permettant de créer des matrices particulières : matrice transposée ; matrice diagonale, matrice triangulaire, matrice unitaire, matrice nulle, etc.

Voyons quelques exemples :

- ***ones***(n) : permet de crée une matrice carrée à *nxn* éléments initialisés à « 1 »
- ***ones*** (n,p) : permet de crée une matrice à *n* lignes et *p* colonnes d'élément initialisés à « 1 »
- ***zeros***(n) : permet de crée une matrice carrée à *nxn* éléments initialisés à « 0 »
- ***zeros*** (n,p) : permet de crée une matrice à *n* lignes et *p* colonnes d'élément initialisés à « 0 »
- ***eye***(n) : permet de crée une matrice carrée unitaire à *nxn*.
- ***eye*** (n,p) : permet de crée une matrice « unité » à *n* lignes et *p* colonnes .
- ***diag(vec)*** : Appliquée à un vecteur-ligne ou vecteur-colonne ***vec***, cette fonction retourne une matrice carrée dont la diagonale principale porte les éléments du vecteur ***vec*** et les autres éléments sont égaux à "0".
- ***diag(vec)*** : Appliquée à une matrice, elle retourne, dans un vecteur-colonne, les éléments de la diagonale de cette matrice.
- ***repmat(mat1,M,N)*** : Renvoie une matrice mat2 formée à partir de la matrice mat1 dupliquée en "tuile" M fois verticalement et N fois horizontalement.

Encore d'autres fonctions pour s'informer !

Octave mis à notre disposition des fonctions permettant de nous donner les dimensions d'un scalaire, d'un vecteur ou d'une matrice. On peut aussi avoir leur nombre de ligne(s) et leur nombre de colonne(s) et bien d'autres informations encore !

Voyons quelques exemples :

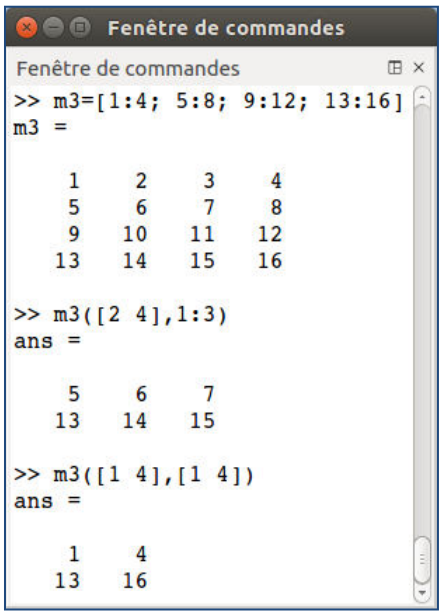
- ***size(var)*** : renvoi le nombre de lignes et de colonnes de la variable ***var*** qui peut être un scalaire (donc 1 ligne x 1 colonne), un vecteur ou une matrice.
- ***size(var, dim)*** : renvoi le nombre d'éléments de la variable ***var*** correspondant à la dimension spécifiée. Par exemple pour une matrice M à 5x7 éléments, l'expression *size(M,1)* va renvoyer 5 (5 lignes) alors que l'expression *size(M,2)* va renvoyer 7.
- ***rows*** et ***columns*** renvoient respectivement le nombre de lignes et le nombre de colonnes d'une matrice.
- ***length(mat)*** : Appliquée à une matrice, cette fonction analyse le nombre de lignes et le nombre de colonnes puis retourne le plus grand de ces 2 nombres (donc identique à max(size(mat))). Cette fonction est par conséquent assez dangereuse à utiliser sur une matrice !
- ***numel(mat)*** : Retourne le nombre d'éléments du tableau mat.

Comment accéder aux éléments d'une matrice ?

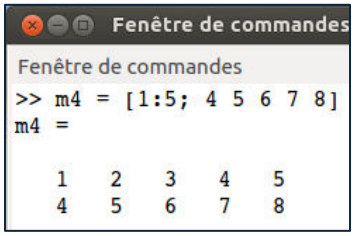
C'est comme en mathématiques : On indique le nom de la matrice et entre parenthèses deux indices, le premier indique le numéro de ligne et le seconde le numéro de colonne. Cela est, bien évidemment, la forme la plus simple pour accéder aux éléments d'une matrice. Cependant, en général, les deux indices sont des vecteurs, le premier se rapportant à la première dimension (les lignes) et le second à la seconde dimension (les colonnes). On peut utiliser, ainsi, toutes les subtilités des séries pour construire ces vecteurs d'indices. Dans la forme simplifiée où l'on utilise « : » à la place du premier vecteur d'indices, cela désignera toutes les lignes ; respectivement si l'on utilise « : » la place du second vecteur d'indices, cela désignera toutes les colonnes.

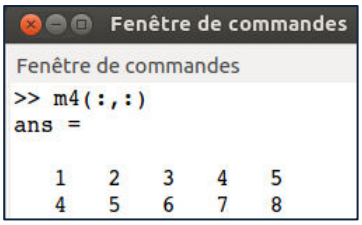
Voyons quelques exemples :

- si l'on a la matrice m3=[1:4; 5:8; 9:12; 13:16]
  m3([2 4],1:3) désigne les éléments des lignes 2 et 4 se trouvant dans les colonnes 1 à3 ce qui retourne [5 6 7 ; 13 14 15]
  m3([1 4],[1 4]) désigne les éléments des lignes 1 et 4 se trouvant dans les colonnes 1 et 4 ce qui retourne [1 4 ; 13 16]

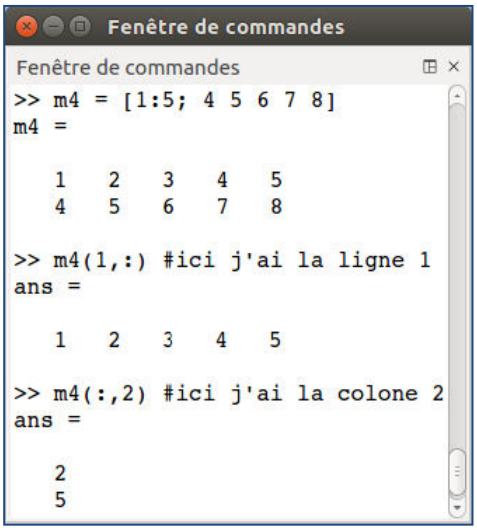


- « : » permet d'indiquer toutes les lignes ou toutes les colonnes :





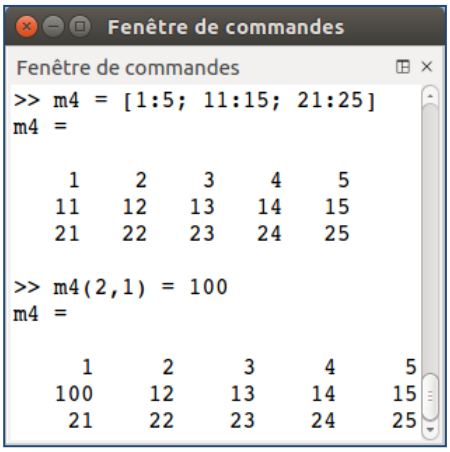
- Je peux considérer uniquement les lignes ou les colonnes :



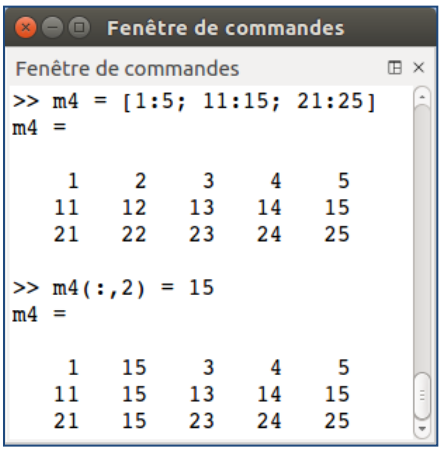
Modifier les éléments d'une matrice :

Il suffit d'indiquer une ou plusieurs cellules de la matrice et leur affecter une valeur.

Exemple 1 : Je vais modifier la valeur de l'élément se trouvant en 2ème ligne et première colonne d'une matrice ***m4*** en y mettant la valeur *100* :

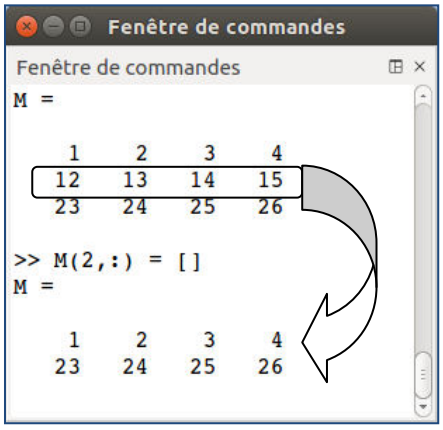


Exemple 2 : Je vais modifier les valeurs de toute la colonne 2 d'une matrice m4 en y mettant la valeur 15 :

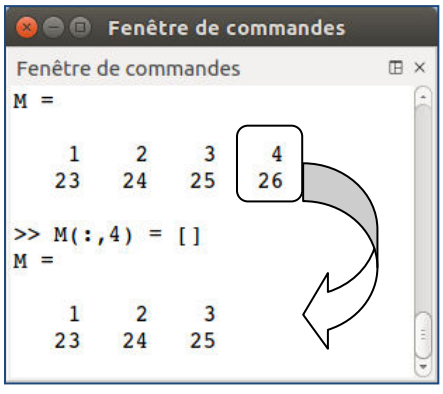


Destruction d'éléments d'une matrice : Il est possible de supprimer des lignes ou des colonnes entières d'une matrice en lui affectant un vecteur vide ([]).

Exemple 1 : je vais supprimer la seconde ligne de la matrice M :



Exemple 2 : je vais supprimer la quatrième colonne de la matrice M :



Attention : Vous ne pouvez pas supprimer physiquement une cellule individuelle d'une matrice. Cela n'est pas possible car vous allez remettre en cause la définition même d'une matrice (le nombre d'éléments doit être identique pour toutes les colonnes d'un coté et pour toutes les lignes de l'autre).

Opérations sur les matrices : Diverses opérations peuvent être appliquées aux matrices :

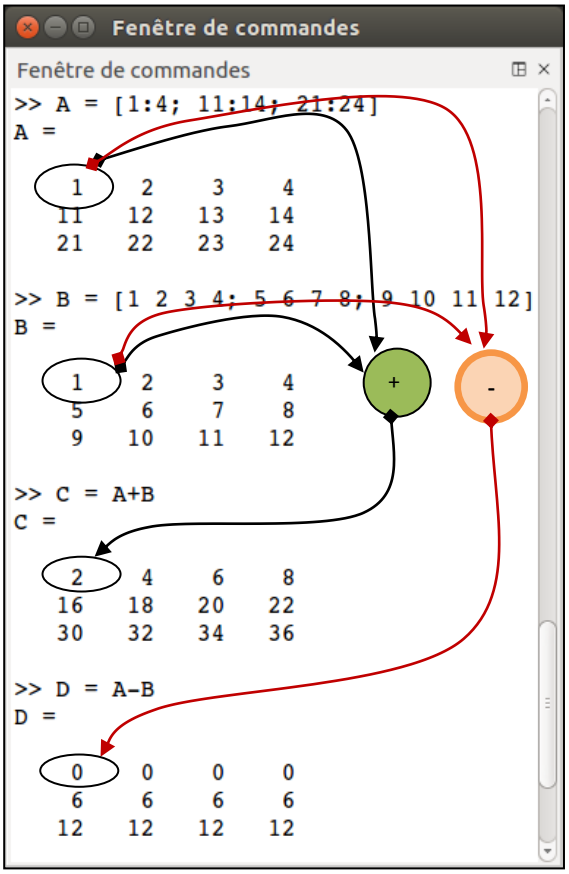
Addition et soustraction
Multiplication par un scalaire
Transposition
Multiplication élément par éléments
Produit scalaire
Etc.

Addition et soustraction : C'est comma en mathématique. Ayant deux matrices A et B ayant toutes les deux n lignes et p colonnes, on peut calcul la somme C = A+B de sorte que chaque élément de C est la somme des éléments de A et de B ayant les mêmes indices.

De la même façon, on peut calculer la soustraction de deux matrice D = A-B de sorte que les éléments de D est la différence des éléments de A et de B ayant les mêmes indices.

C = A+B : cij = aij + bij, D = A-B : dij = aij - bij, Avec i allant de 1 1àn et j allant de 1 à p

Voici un exemple avec Octave :

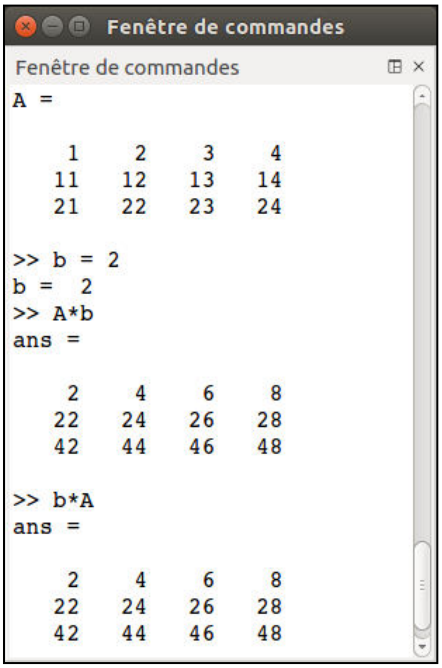


Attention : L'addition et la soustraction matricielles ne sont possibles que sur des matrices ayant les mêmes dimensions !

Multiplication par un scalaire : Soit *A* une matrice *NxP*, et *b* un scalaire. : $A . b = b . A = C$

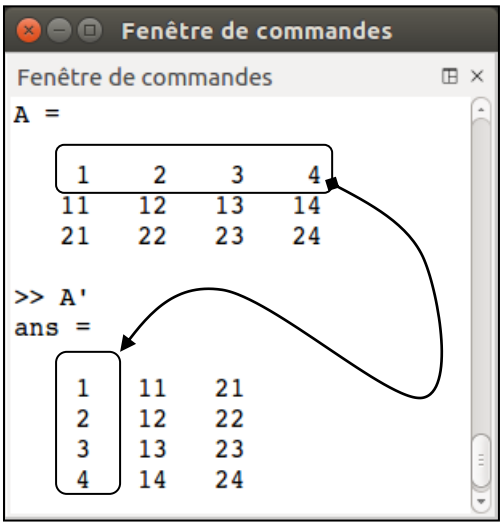
C étant une matrice ayant les mêmes dimensions que *A* (*NxP*) et dont chaque éléments $c_{ij} = b.a_{ij}$.

Voyons un exemple avec Octave :



Transposition : Comme pour les vecteurs, on peut calculer la transposée d'une matrice. Soit *A* une matrice à *n* lignes et *p* colonnes. La matrice transposée de A est notée ***A'*** est composée des éléments $a'_{ij}=a_{ji}$. En d'autres termes, les lignes de *A* deviennent les colonnes de *A'* et inversement, les colonnes de A deviennent les lignes de *A'*.

Voyons un exemple avec Octave :



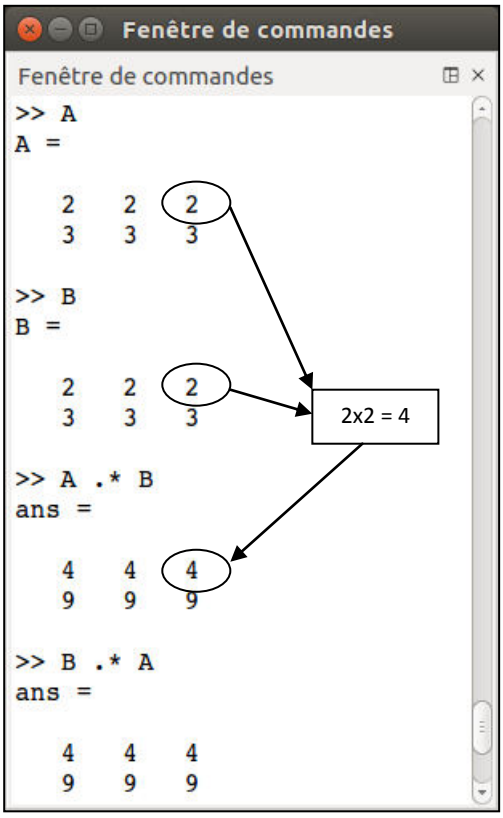
Multiplication terme à terme : Il est tout à fait possible d'effectuer le produit terme à terme entre 2 matrices *A* et *B* de même dimensions.

$A = [a_{ij}]$ et $B = [b_{ij}]$ avec *i* allant de 1 à *n* et *j* allant de 1 à *p*.

Alors pour réaliser le produit terme entre *A* et *B*, on utilise l'opérateur « **.*** » dans Octave.

C = A .* B = B .* A

Voyons un exemple avec Octave :



Produit scalaire :

A suivre …

QCM 3 – Manipuler des vecteurs et des matrices

1 - Scalaires

Q1 - Les nombres (simple, double, complex et entiers) sont :

- ☐ Des scalaires
- ☐ Des vecteurs
- ☐ Des matrices

Q2 – Un scalaire est une matrice particulière une ligne et une colonne

- ☐ Vrai
- ☐ Faux

Q3 – La fonction ***isscalar***() permet d'indiquer

- ☐ Si une variable est un vecteur
- ☐ Si une variable est un nombre
- ☐ Si une variable est une matrice
- ☐ Si une variable est un scalaire

2 - Séries

Q4 - La commande « x = 1 :8 » permet de :

- ☐ Créer une matrice composée de 8 éléments
- ☐ Créer un vecteur-ligne composé des nombre entier 1 à 8
- ☐ Créer un vecteur-ligne composé des nombres réels « double »
- ☐ Créer un vecteur-colonne composé des nombres réels « double »

Q5 - La commande « x = 10 :1 » permet de :

- ☐ Créer un vecteur-ligne composé d'une ligne et de zéro colonne (donc vecteur vide)
- ☐ Créer un vecteur-ligne composé d'une ligne et d'une colonne
- ☐ Créer un vecteur-ligne composé des scalaires allant de 10 à 1

Q6 - La commande « x = 10 :-1 :2 » permet de :

- ☐ Créer un vecteur-ligne composé d'une ligne et de zéro colonne (donc vecteur vide)
- ☐ Créer un vecteur-ligne composé des scalaires allant de 10 à 2
- ☐ Créer un vecteur-ligne composé des scalaires allant de 2 à 10

Q7 - La commande « x = 1 : 2 : 6 » permet de créer un vecteur-ligne composé des scalaires suivants:

- ☐ 1, 3, 5
- ☐ 5, 3, 1
- ☐ 1, 2, 3, 4, 5, 6
- ☐ 6, 5, 4, 3, 2, 1

Q8 - La commande « x = 10 : -2 : 6 » permet de créer un vecteur-ligne composé des scalaires suivants:

- ☐ 6, 8, 10
- ☐ 8, 6, 10
- ☐ 6, 8, 10
- ☐ 6, 7, 8, 9, 10

Q9 - La commande « x = 1 : 0.5 : 3 » permet de créer un vecteur-ligne composé des scalaires suivants:

- ☐ 1.0000 1.5000 2.0000 2.5000 3.0000
- ☐ 1.0000 2.0000 3.0000
- ☐ 2.0000 3.0000 1.0000

Q10 - Lorsqu'on connait la valeur de départ et la valeur finale et l'on souhaite générer un nombre bien précis de valeurs entre ces deux bornes, quelle est la fonction Octave que vous utiliserez ?

.....

Q11 – Donnez la commande permettant de générer 100 valeurs d'un axe des abscisses X allant de -5 à +5

.....

3 – Les vecteurs

Q12 – Un vecteur-ligne est composé :

- ☐ De plusieurs lignes mais une seule colonne
- ☐ De plusieurs colonnes mais une seule ligne
- ☐ De plusieurs lignes et de plusieurs colonnes

Q13 – Un vecteur-colonne est composé :

- ☐ De plusieurs lignes mais une seule colonne
- ☐ De plusieurs colonnes mais une seule ligne
- ☐ De plusieurs lignes et de plusieurs colonnes

Q14 – Pour créer un vecteur-ligne on utilise comme séparateur des valeurs:

- ☐ L'espace
- ☐ La tabulation
- ☐ La virgule
- ☐ Le point-virgule
- ☐ La touche « ENTER »

Q15 – Pour créer un vecteur-colonne on utilise comme séparateur des valeurs:

- ☐ L'espace
- ☐ La tabulation
- ☐ La virgule
- ☐ Le point-virgule
- ☐ La touche « ENTER »

Q16 – En utilisant l'espace comme séparateur, donnez la commande permettant de créer le vecteur-ligne suivant : (1, 5, 6, 10) :

.....

Q17 – En utilisant la virgule comme séparateur, donnez la commande permettant de créer le vecteur-ligne suivant : (1, 5, 6, 10)

.....

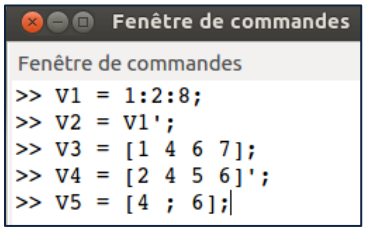
Q18 – La commande permettant de créer le vecteur-ligne suivant : X = (1, 5, 6, 10) est :

- ☐ X = [1 5 6 10]
- ☐ X = [1, 5, 6, 10]
- ☐ X = [1 ; 5 ; 6 ; 10]
- ☐ X = [1 5 ; 6 10]
- ☐ X = [1 : 5 : 6 : 10]

Q19 – Donnez la commande permettant de créer le vecteur-colonne suivant : Y= (5, 3, 6, 10)

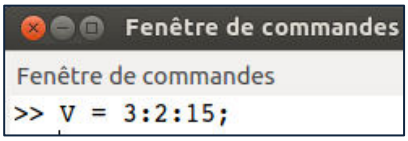
- ☐ Y = [5 3 6 10]
- ☐ Y = [5, 3, 6, 10]
- ☐ Y = [5 ; 3 ; 6 ; 10]
- ☐ Y = [5 3 ; 6 10]
- ☐ Y = [5 : 3 : 6 : 10]

Q20 – A l'issue des commandes suivantes :



- ☐ V1 est un vecteur-ligne
- ☐ V1 est un vecteur-colonne
- ☐ V2 est un vecteur-ligne
- ☐ V2 est un vecteur-colonne
- ☐ V3 est un vecteur-ligne
- ☐ V3 est un vecteur-colonne
- ☐ V4 est un vecteur-ligne
- ☐ V4 est un vecteur-colonne
- ☐ V5 est un vecteur-ligne
- ☐ V5 est un vecteur-colonne

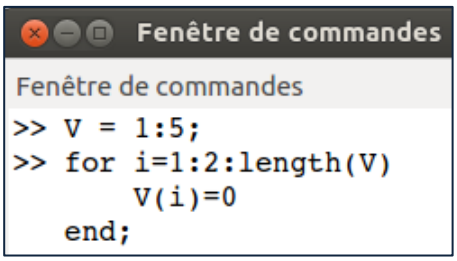
Q21 – On suppose que l'on a crée un vecteur V comme suit :



Reliez les commandes ci-dessus aux valeurs quelles vont générer

V(2)	5
size(V)	1 7
length(V)	7
V(1:3)	3 5 7
V(6:-1:3)	13 11 9 7
V(end)	15
V(end:-2:1)	15 11 7 3
V(1:2:5)	3 7 11

Q22 – Soit la séquence de commandes suivantes :



A l'issue de ces commandes, V contiendra :

- ☐ 0 2 0 4 5
- ☐ 0 2 3 4 5
- ☐ 0 2 0 4 0
- ☐ 1 2 3 4 5

Q23 – Donnez la commande permettant de mettre la valeur 15 dans le vecteur V dans les positions : 1, 5 et 7

.....

Q24 – Donnez la commande permettant de supprimer les valeurs situées dans les positions paires d'un vecteur V

. .....

4 – Les matrices

Q25 – Quelles est la commande octave me permettant de créer la matrice ***M*** suivante?

$$M = \begin{pmatrix} -2 & -1 & 0 \\ 41 & 13 & 3 \end{pmatrix}$$

- ☐ M = [-2, -1, 0; 41, 13, 3] ;
- ☐ M = [-2 -1 0 ; 41 13 3] ;
- ☐ M = [-2 ; -1 ; 0 ; 41 ; 13 ; 3];
- ☐ M = [-2 -1 0 41 13 3];

Q26 – Quelles est la commande octave me permettant de créer la matrice ***M1*** suivante?

$$M1 = \begin{pmatrix} 1 & 8 \\ 4 & 10 \\ 5 & 7 \end{pmatrix}$$

- ☐ M1 = [1 , 8 , 4 , 10 , 5 , 7] ;
- ☐ M1 = [1 8 4 ; 10 5 7] ;
- ☐ M1 = [1 , 8 , 4 ; 10 5 7] ;
- ☐ M1 = [1 ; 8 ; 4 , 10 ; 5 ; 7] ;

Q27 – Quelles est la commande octave me permettant de créer la matrice ***M2*** suivante?

$$M2 = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 10 & 8 & 6 & 4 \\ 10 & 12 & 14 & 16 \end{pmatrix}$$

- ☐ M2 = [1 : 2 : 8 ; 10 : -2 : 3 ; 10 : 2 : 17] ;
- ☐ M2 = [1,3,5,7 ; 10,8,6,4 ; 10,12,14,16] ;
- ☐ M2 = [1 3 5 7 ; 10 8 6 4 ; 10 12 14 16] ;
- ☐ M2 = [1 : 2 : 8 ; 10 : -2 : 3 ; 10 12 14 16] ;

Q28 – J'ai crée deux vecteurs V1 et V2comme suit :

```
Fenêtre de commandes
>> V1 = 1:5;
>> V2 = [11:15]' ;
```

Que va contenir M3 à l'issue de la commande suivante :

M3 = [V1, V2]

- ☐ Une matrice 1x10
- ☐ Une matrice 10x1
- ☐ Une matrice 10 x 10
- ☐ Ça renvoi une erreur

Q29 – Que va contenir M4 à l'issue des commandes suivantes :

```
Fenêtre de commandes
>> V1 = 1:5;
>> V2 = [11:15] ;
>> M4 = [V1; V2];
```

- ☐ Une matrice 2x10
- ☐ Une matrice 2x5
- ☐ Une matrice 5x2
- ☐ Une matrice 10x2
- ☐ Ça renvoi une erreur

Q30 – Que va contenir M5 à l'issue des commandes suivantes :

```
Fenêtre de commandes
>> V1 = 1:5;
>> V2 = [11:15] ;
>> M5 = [V1  V2];
```

- ☐ Une matrice 2x10
- ☐ Une matrice 1x10
- ☐ Une matrice 10x1
- ☐ Une matrice 10x2
- ☐ Ça renvoi une erreur

Q31 – La concaténation de vecteurs-lignes de dimensions différentes est possible

- ☐ Vrai
- ☐ Faux

Q32 – La concaténation de vecteurs-colonnes de dimensions différentes est possible

- ☐ Vrai
- ☐ Faux

Q33 – La concaténation horizontale de vecteurs-colonnes de dimensions différentes est possible

- ☐ Vrai
- ☐ Faux

Q34 – La concaténation horizontale de vecteurs-lignes de dimensions différentes est possible

- ☐ Vrai
- ☐ Faux

Q35 – La concaténation verticale de vecteurs-colonnes de dimensions différentes est possible

- ☐ Vrai
- ☐ Faux

Q36 – La concaténation verticale de vecteurs-lignes de dimensions différentes est possible

- ☐ Vrai
- ☐ Faux

Q37 –Pour pouvoir concaténer deux matrices horizontalement

- ☐ il faut qu'elles aient les mêmes dimensions
- ☐ il faut qu'elles aient le même nombre de colonnes
- ☐ il faut qu'elles aient le même nombre de lignes
- ☐ est toujours possibles
- ☐ est impossible

Q38 –Pour pouvoir concaténer deux matrices verticalement

- ☐ il faut qu'elles aient les mêmes dimensions
- ☐ il faut qu'elles aient le même nombre de colonnes
- ☐ il faut qu'elles aient le même nombre de lignes
- ☐ est toujours possibles
- ☐ est impossible

Q39 – Donnez la commande Octave me permettant de créer une matrice unité 5x5

Reliez les commandes ci-dessus aux objets qui seront retournés

zeros(5)	Matrice 5x5 remplie de « 0 »
ones(5,5)	Matrice 5x5 remplie de « 1 »
eye(5)	Matrice unitaire 5x5
rand(5)	Matrice 5x5 remplie de valeurs tirées au hasard entre « 0 » et « 1 »
diag(ones(5))	Matrice 5x1 composée de « 1 »
diag(ones(5)) ’	Matrice 1x5 composée de « 1 »
rand(1,5)	Matrice 1x5 composée de valeurs tirées au hasard entre 0 et 1
rand(4,1)	Matrice 4x1 composée de valeurs tirées au hasard entre 0 et 1
rows(M)	Renvoie le nombre de lignes de la matrice M
columns(M)	Renvoie le nombre de colonnes de la matrice M
size([1 :5 ;12 :16])	2 5
length([1 :5 ;12 :16])	5
numel([1 :5 ;12 :16])	10

Q40 – Je suppose que j’ai exécuté les commandes Octave suivantes :

```
Fenêtre de commandes
>> A = [1 2; 3 4; 5 6];
>> B = [1 2 3; 4 5 6];
```

Parmi les commandes octave ci-dessous, indiquez celles qui sont valides:

- ☐ C = A’ * B
- ☐ C = A * B
- ☐ C = B * A
- ☐ C = B’ * A

Q41 – Je suppose qu’on a crée la matrice A suivante :

$$A = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$$

Parmi les commandes ci-dessous, indiquez celles qui permettent d’obtenir la matrice B suivante :

$$B = \begin{bmatrix} 16 & 2 \\ 5 & 11 \\ 9 & 7 \\ 4 & 14 \end{bmatrix}$$

- ☐ B = A(: , 0 : 2)
- ☐ B = A(0 : 4, 0 : 2)
- ☐ B = A(: , 1 : 2)
- ☐ B = A(1 : 4 , 1 : 2)

Q42 – Je suppose que j’ai crée une matrice A à 10x10 de nombres réels compris entre 0 et 1. Je suppose aussi que j’ai crée un vecteur-colonne x lui aussi composé de réels compris entre 0 et 1.

```
Fenêtre de commandes
>> A = rand(10,10);
>> x = rand(10,1);
```

Votre camarade veut calculer le produit ***Ax*** et a écrit le code suivant :

```
v = zeros(10, 1);
for i = 1:10
  for j = 1:10
    v(i) = v(i) + A(i, j) * x(j);
  end
end
```

Quels commande octave utiliseriez-vous pour faire ce qu’à fait votre camarade ?

- ☐ v = A * x ;
- ☐ v = Ax ;
- ☐ v = sum(A*x) ;
- ☐ v = x’ * A ;

Q43– Je suppose que j’ai crée deux vecteur colonnes v et w comme suit :

```
Fenêtre de commandes
>> v = [1:7]';
>> w = [11:17]';
```

Prenons le code suivant :

```
z = 0;
for i = 1:7
  z = z + v(i) * w(i);
end
```

Quels commande octave utiliseriez-vous pour réalisez ce qui fait le code ci-dessus ?

- ☐ z = w’ * v ;
- ☐ z = v’ * w ;
- ☐ z = v * w’ ;
- ☐ z = v * w ;

TP3 – Manipuler des vecteurs et des matrices

Exo 1 : séries (opérateur « : »)

Démarrer Octave puis tapez « **x = -1:0.1 :1 »** puis exécuter les commandes suivantes :

1	sqrt(x)
2	sin(x)
3	tan(x)
4	x^2
5	x.^3
6	plot(x, cos(x.^3))
7	plot(x, cos(x.^2))

Questions :

Q1 – Que contient la variable « **x** » ?

- ☐ un réel
- ☐ un complex
- ☐ un vecteur
- ☐ un scalaire
- ☐ une matrice

Q2 – Dans la commande « **x = -1 : 0.1 : 1 »** la valeur « -1 » représente ?

- ☐ le premier élément du vecteur généré
- ☐ le dernier élément du vecteur généré
- ☐ le pas de progression d'une valeur à l'autre du vecteur généré

Q3 – Dans la commande « **x = -1 : 0.1 : 1 »** la valeur « 1 » représente ?

- ☐ le premier élément du vecteur généré
- ☐ le dernier élément du vecteur généré
- ☐ le pas de progression d'une valeur à l'autre du vecteur généré

Q4 – Dans la commande « **x = -1 : 0.1 : 1 »** la valeur « 0.1 » représente ?

- ☐ le premier élément du vecteur généré
- ☐ le dernier élément du vecteur généré
- ☐ le pas de progression d'une valeur à l'autre du vecteur généré

Q5 – Que contient la variable «sin(**x)** » ?

- ☐ un réel
- ☐ un complex
- ☐ un vecteur
- ☐ un scalaire
- ☐ une matrice

Q6 – La fonction «sin » nécessite un paramètre exprimé en radian ?

Indication : utilisez la commande « help sin»

☐ Vrai ☐ Faux

Q7 – En ligne 6?

- ☐ ***plot*** est une fonction qui affiche la courbe $f(x) = cos(x^3)$
- ☐ ***plot*** est une fonction qui affiche la courbe $f(x) = cos(x^2)$
- ☐ ***plot*** est une fonction qui affiche la courbe $f(x) = ax + b$
- ☐ x est un scalaire
- ☐ x est une matrice
- ☐ x est un vecteur-ligne
- ☐ x est un vecteur-colonne

Exo2 : Création et manipulation de vecteurs

Q8 : En utilisant l'espace comme indicateur de colonnes, créer un vecteur-ligne **V1** composé des valeurs suivantes : V1 = (0, 3, 6, 9, 12)

.....

Q9 : En utilisant l'opérateur « : » (générateur de séquences), créer un vecteur-ligne **V2** composé des valeurs suivantes : V2 = (0, 3, 6, 9, 12).

Indication : le pas doit être positif.

.....

Q10 : En utilisant l'opérateur « : » (générateur de séquences), créer un vecteur-ligne **V2** composé des valeurs suivantes : V2 = (0, 3, 6, 9, 12).

Indication : le pas doit être négatif.

.....

Q11 : Soit le vecteur V1 = (0, 3, 6, 9, 12). Donnez la commande Octave permettant d'afficher uniquement les valeurs de V1 situées dans les 3 premières positions.

Indication : Utilisez l'opérateur « : » !

.....

Q12 : Soit le vecteur V1 = (0, 3, 6, 9, 12). Donnez la commande Octave permettant d'afficher uniquement les valeurs de V1 situées dans les positions impaire en commençant de « 1 ».

Indication : Utilisez l'opérateur « : » !

.....

Q13 : Soit le vecteur V1 = (1, 3, 6, 9, 12). Donnez la commande Octave permettant de modifier la valeur 3 par 5.

.....

Exo3 : Création et manipulation de matrices

Q14 : Donnez la commande octave permettant de créer la matrice ***M1*** suivante :

$$M1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

.....

Q15 : Donnez la commande octave permettant de créer la matrice ***M2*** suivante :

$$M2 = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

.....

Q16 : Donnez la commande octave permettant de créer la matrice ***M3*** suivante :

$$M3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

.....

Q17 : En utilisant l'opérateur « : », donnez la commande octave permettant de créer la matrice ***M4*** suivante :

$$M4 = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

.....

Q18 : Donnez la commande octave permettant de créer une matrice ***M5*** composée de 2 lignes et 3 colonnes de réels tirées au hasard entre 0 et 1 :

.....

Q19 : En vous servant de la matrice ***M5*** de la question précédente, donnez la commande octave permettant de créer une matrice ***M6*** composée de 2 lignes et 3 colonnes de réels tirées au hasard entre 4 et 5 :

.....

Q20 : En vous servant de la matrice ***M5*** de la question précédente, donnez la commande octave permettant de créer une matrice ***M7*** composée de 2 lignes et 3 colonnes de réels tirées au hasard entre 4 et 10 :

.....

Q21 : Donnez la commande octave permettant de créer la matrice ***M8*** qui est la transposée de ***M7*** :

.....

Q22 : Donnez la commande octave permettant de rendre dans la variable ***n*** le nombre de lignes de la matrice ***M8*** :

.....

Q23 : Donnez la commande octave permettant de rendre dans la variable ***p*** le nombre de colonnes de la matrice ***M8*** :

.....

Q24 : Donnez la commande octave permettant de rendre dans un vecteur ***d*** les dimensions (nombre de lignes et de colonnes) de la matrice ***M8*** :

.....

Q25 : Donnez la commande octave permettant d'extraire la diagonale de la matrice ***M8*** dans un vecteur ***Vd*** :

.....

Q26 : Soit la matrice ***M9*** suivante :

$$M9 = \begin{pmatrix} 41 & 10 & 70 \\ 10 & 10 & 45 \\ 33 & 11 & 31 \\ 11 & 20 & 37 \\ 70 & 31 & 28 \\ 60 & 40 & 54 \end{pmatrix}$$

A - Donnez la commande octave permettant d'extraire dans une matrice ***M10*** les lignes 2 et 3 de ***M9***

.....

B - Donnez la commande octave permettant d'extraire dans une matrice ***M10*** les lignes 1 et 3 de ***M9***

.....

C - **M9(end,end)** représente quel élément de ***M9***

.....

D - **M9(end,end)** et **M9(end)** représente le même élément : ☐Vrai ☐Faux

Q27 : Dans la commande « **X = M9 <20** », la variable « X » est une matrice ayant les mêmes dimensions que ***M9*** :

- ☐ ***Xij*** =0 implique ***M9ij*** <20
- ☐ ***Xij*** =1 implique ***M9ij*** <20
- ☐ ***M9(X)*** indique un vecteur composée des éléments de ***M9*** supérieurs à 20
- ☐ ***M9(X)*** indique un vecteur composée des éléments de ***M9*** inférieurs à 20

Q28 : Donnez la commande octave permettant de faire le produit matriciel de M1 x M2 :

.....

Q29 : Donnez la commande octave permettant de faire le produit élément par élément de M1 x M2 :

.....

Q29 : Donnez la commande octave permettant de calculer la somme de tous les éléments de la matrice M4 :

.....

30 : Soient les commandes suivantes ;

numéros	Commandes
1	M = ones(5)*77; Ou M = zeros(5); M(:) = 77;
2	M = zeros(5); Ou M(5,5) = 0;
3	M = ones(3,5);
4	M = ones(5);
5	M = ones(3,5)*77; Ou M = zeros(3,5); M(:) = 77;
6	M = zeros(3,5); Ou M(3,5) = 0;
7	M = eye(3,5);
8	M(3,5) = 1;
9	M = eye(5); M = M(:,end:-1:1)
10	M = eye(5);

En vous basant sur le tableau des commandes ci-dessus, indiquez lesquelles pourront générer les matrices suivantes :

Numéros des commandes	Matrices
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	77 77 77 77 77 77 77 77 77 77 77 77 77 77 77
	77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77 77
	1 0 0 0 0 0 1 0 0 0 0 0 1 0 0
	1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1
	0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Chapitre 4 – Véritable programmation

Octave permet à l'utilisateur de définir ses propres fonctions et de nouveaux scripts étendant ainsi les fonctionnalités offertes. Il s'agit d'un véritable langage de programmation ! Ici, on présentera les possibilités de programmation d'Octave. A l'issue de ce chapitre, on s'attend à ce que vous serez capables d'écrire de véritables scripts en vous servant :

- de l'interaction avec des utilisateurs
- des structures de contrôles établissant la logique de vos scripts
- de fonctions
- de scripts
- d'interfaces graphiques
- de fichiers de données

1 – Interaction

Il s'agit des commandes permettant à un utilisateur de communiquer avec un programme par le biais de périphérique comme le clavier, l'écran, la carte son etc.

Affichage de texte et de variables

La commande « disp » et « printf ou printf » permettent d'afficher des variables ou du texte.

☞ disp(variable) ou disp(chaine)

Affiche la chaine de caractères ou le contenu de la variable spécifiée. Les nombres sont formatés conformément à la commande « format ».

Exemple :

```
Fenêtre de commandes
>> M = [ 2 4; 7 9];
>> disp('la matrice M est : '), disp(M)
la matrice M est :
   2   4
   7   9
```

☞ pintf('format', liste de variables) ou
fpintf('format', liste de variables)

Affiche de façon formatée, la ou les variables spécifiées. Cette fonction ainsi que la syntaxe de formatage est reprise du langage C.

Exemple :

```
Fenêtre de commandes
>> temps = 456; t = 'Voici le temps mis : ';
>> printf("%s=%6.2f\n",t,temps)
Voici le temps mis : =456.00
```

Dans l'exemple ci-dessus, j'ai utilisé la commande « ***printf*** » exactement comme dans le langage C. Dans les paramètres de cette fonction (dans notre exemple), j'ai trois choses :

- "%s=%6.2f\n" : qui est la partie formatage. Ici j'ai indiqué qu'il faut afficher une première variable chaine puis le caractère « = » puis un nombre réel sur 6 chiffres dont 2 décimales
- t : est la première variable que sera affiché, elle est de types chaine
- temps : est ma deuxième variable

Entrée d'informations au clavier

C'est la commande « input » qui permet à l'utilisateur d'introduire des données depuis le clavier. Cette commande peut être utilisée deux façons :

- input("message", "s")
- input("message")

La première forme permet d'afficher un message à l'écran et récupère la saisie de l'utilisateur sous forme d'une chaine de caractères

La seconde forme permet à l'utilisateur de saisir des données de types quelconques comme des scalaires, des vecteurs ou des matrices.

Exemple1 : Lecture d'une chaine de caractères

```
Fenêtre de commandes
>> uneChaine = input("donnez votre nom : ", 's')
donnez votre nom : Zinneddine Zidane
uneChaine = Zinneddine Zidane
>> class(uneChaine)
ans = char
```

Dans cet exemple, on permet d'introduire une chaine de caractère au clavier.

Exemple2 : Lecture d'un scalaire

```
Fenêtre de commandes
>> unReel = input("donnez x : ")
donnez x : 178.5
unReel =  178.50
>> class(unReel)
ans = double
>>
```

Dans cet exemple, on permet d'introduire un réel au clavier.

Exemple 3 : Lecteur d'un vecteur

```
Fenêtre de commandes
>> unVecteur = input("donnez les coordonnées de x : ")
donnez les coordonnées de x : [14, 15, 16, 17]
unVecteur =

   14   15   16   17
```

Exemple 4 : lecture d'une matrice

```
Fenêtre de commandes
>> M = input("donnez les données de y : ")
donnez les données de y : [14 15 16; 14 15 78]
M =

   14   15   16
   14   15   78
```

Attention aux erreurs de saisie :

```
Fenêtre de commandes
>> M = input("donnez les données de y : ")
donnez les données de y : [12, 14; 15 ; 16]
error: vertical dimensions mismatch (1x2 vs 1x1)
```

Dans l'exemple ci-dessus, l'utilisateur a saisi les données d'une matrice, mais il y a eu un problème dans les dimensions. Il a indiqué une première ligne composée de 2 colonnes, mais les deux lignes suivantes ne sont composées que d'une seule colonne, ce qui est incohérent.

Il est tout à fait possible de dire à octave d'attendre jusqu'à ce que l'utilisateur tape sur une touche pour poursuivre le traitement. Pour cela on utilise la commande « **pause** ». On peut même lui dire d'attendre un certain temps (en secondes) avant de poursuivre, pour cela on utilise « **pause(secondes)** » avec « secondes le nombre de secondes d'attente avant de poursuivre le traitement.

2 – Structures de contrôle

Comme tous les autres langages de programmation, Octave offre des commandes pour effectuer des traitements conditionnels ou réplétifs.

Les conditions

La base du contrôle de flux des commandes dans un programme est l'évaluation de conditions. Une condition est une expression pouvant être vraie ou fausse. Elle est composée grâce à des opérateurs de relation et des opérateurs logiques.

Les opérateurs relationnels

Ils permettent de faire des tests numériques en construisant des "expressions logiques", c'est-à-dire des expressions retournant les valeurs vrai ou faux

Opérateur	Description
==	Test d'égalité
~= !=	Test d'inégalité
<	Tes d'infériorité
>	Test de supériorité
>=	Test de supériorité ou égalité
<=	Tes d'infériorité ou égalité

A noter que vous pouvez utiliser des fonctions à la place de ces opérateurs de relation :

Opérateur	Fonction correspondant
==	eg
~= !=	ne
<	lt
>	gt
>=	ge
<=	le

Les opérateurs logiques

Les opérateurs logiques ont pour arguments des expressions logiques et retournent les valeurs logiques vrai (1) ou faux (0).

Opérateur	Description
~ expression	Négation logique
Exp1 & Exp2	Et logique entre Exp1 et Exp2
Exp1 && Exp2	Et logique entre Exp1 et Exp2, mais plus efficace que le simple &
Exp1 | Exp2	Ou logique entre Exp1 et Exp2
Exp1 || Exp2	Ou logique entre Exp1 et Exp2, mais plus efficace que le simple |

A noter que vous pouvez utiliser des fonctions à la place de ces opérateurs logiques :

Opérateur	Fonction correspondant
Exp1 & Exp2	and(Exp1, Exp2)
Exp1 | Exp2	Or(Exp1, Exp2)

Il existe aussi une fonction permettant de réaliser le ou exclusif entre plusieurs expressions :
xor(Exp1, Exp2, ...).

La commande for

Syntaxe :

```
for var = expression
        Instructions
endfor
```

L'expression peut être un vecteur ou une matrice.

Exemple 1 : Calcul des éléments de la suite de *Fibonnacci* pour n = 10 :

```
Fenêtre de commandes
>> n = 10;
>> u = ones(1, n);
>> for i = 3:n
       u(i) = u(i-1) + u(i-2);
   endfor;
>> disp("notre suite de fibonacci est : ") ; disp(u)
notre suite de fibonacci est :
    1    1    2    3    5    8   13   21   34   55
```

Dans l'exemple ci-dessus on a parcouru une série (les éléments du vecteur 3 à n avec n = 10).

Exemple 2 : Parcours des éléments d'une matrice

```
Fenêtre de commandes
>> M = [1,2,3;4,5,6];
>> for e = M
       disp(e)
   endfor
   1
   4
   2
   5
   3
   6
```

La commande *while*

Elle permet de répéter un traitement tant qu'une condition est vérifiée.

Exemple 1 : Je veux prendre au hasard une valeur comprise entre 0.2 et 1. Je sais que la fonction rand() me renvoi une valeur tirée au hasard entre 0 et 1. Je vais utiliser une boucle tant que la valeur renvoyée par rand() est supérieure à 0.2 je refais le tirage au sorte et je ne m'arrête que lorsque la valeur renvoyée par rand() est comprise entre 0 et 0.2.

```
Fenêtre de commandes
>> x = rand();
>> while x >0.2
       x = rand()
   end
x =   0.079425
```

La commande if

Syntaxe :

```
if condition 1
  Instructions si condition 1 vraie
elseif condition 2
  Instructions si condition 1 vraie
else
  Instructions si toutes les conditions fausses
end
```

Exemple :

```
Fenêtre de commandes
>> age = input("donnez votre âge : ")

donnez votre âge : 12
age =  12
>> if age < 12
      disp("vous êtes un enfant")
   elseif age < 18
      disp("vous êtes un adolescent")
   elseif age < 65
      disp("vous êtes un adulte")
   else
      disp("vous êtes un vieux!")
   end
vous êtes un adolescent
```

Les instructions break et continue

« **break** » permet de rompre un traitement répétitif et d'en sortir immédiatement.

« **continue** » permet de rompre uniquement l'itération courante et de poursuivre sur l'itération suivante.

Exemple :

```
Fenêtre de commandes
>> notes = input("donnez les notes")
donnez les notes[12 14 15 17 23 14]
notes =

   12   14   15   17   23   14

>> for i = 1:length(notes)
       if notes(i)<0 | notes(i)>20
          disp("notes incohérente")
       end
       printf("note %d = %4.2f\n", i, notes(i))
    endfor
note 1 = 12.00
note 2 = 14.00
note 3 = 15.00
note 4 = 17.00
notes incohérente
note 5 = 23.00
note 6 = 14.00
```

Dans l'exemple ci-dessus, on affiche qu'on a rencontré une note incohérente, mais si on décide de ne pas l'afficher on utilisera « continue », mais si on décide de stopper carrément la boucle, on utilisera « break »

```
Fenêtre de commandes
>> notes = input("donnez les notes")
donnez les notes[12 14 15 17 23 14]
notes =

   12   14   15   17   23   14

>> for i = 1:length(notes)
       if notes(i)<0 | notes(
          disp("notes incohére
       end
       printf("note %d = %4.2f\n", i, notes(i))
    endfor
note 1 = 12.00
note 2 = 14.00
note 3 = 15.00
note 4 = 17.00
notes incohérente
note 5 = 23.00
note 6 = 14.00
```

Ajouter « break » ou « continue » ici

La commande switch

La commande ***switch*** permet d'exécuter des instructions selon les valeurs d'une certaine variable.

Syntaxe :

switch expression

case valeur1

instructions si expression vaut valeur1

case valeur2

instructions si expression vaut valeur2

.

.

.

otherwise

instructions sinon

end

Les valeurs de l'expression peuvent être des scalaires ou des chaînes de caractères.

Exemple :

```
Fenêtre de commandes
>> couleurs = ["Vert"; "Rouge"; "Bleu"; "Jaune"];
>> indice = floor(rand()*4)+1;
>> switch couleurs(indice)
     case "Vert"
          disp("j'ai tiré au hazar le Vert")
     case "Rouge"
          disp("j'ai tiré au hazar le Rouge")
     case "Bleu"
          disp("j'ai tiré au hazar le Bleu")
     otherwise
          disp("j'ai tiré au hazar le Jaune")
  end
j'ai tiré au hazar le Jaune
```

Dans l'exemple ci-dessus :

- ligne 1 : je défini un vecteur-colonne composé de 4 chaines de caractères représentant des couleurs
- Ligne 2 : je tire au hasard un nombre compris entre 1 et 4 (la fonction rand me tire au hasard un nombre entre 0 et 1, je le multiplie par 4 pour avoir un nombre entre 0 et 4.
- Le ***switch*** me permet selon la valeur de l'indice d'afficher la couleur qui a été choisie au hasard !

3 – Fonctions

Les fonctions permettent de réutiliser le code et évite ainsi les redondances. Dans octave, c'est la commande « ***function*** » qui permet de déclarer une fonction.

Les fonctions de façon générale sont identifiées par un nom, ont besoins de zéro et de plusieurs paramètres et peuvent renvoyer des valeurs (scalaires, vecteurs ou matrices par exemple).

Syntaxe :

function OUT = nomFonction(IN)

  Corps de la fonction

Avec :

- OUT : les paramètres de sortie pouvant être vide, un scalaire, un vecteur, une matrice, …
- IN : les paramètres d'entrée pouvant être vide, un scalaire, un vecteur, une matrice, …
- Le corps de la fonction est composé d'instructions Octave.

Exemple 1 : Fonction renvoyant la factorielle d'un nombre.

```
Fenêtre de commandes
>> function f = fact(n)
     i = 1;
     f = 1;
     for i = 1:n
         f = f*i;
     endfor
   endfunction
>> fact(5)
ans =  120
```

Remarque : le programme ci-dessus ne fonctionne pas bien car la variable n est de type réel. Si l'utilisateur introduit une valeur qui n'est pas entière, la fonction ne donnera pas un bon résultat !

Exemple 2 :

```
Fenêtre de commandes
>> M = round(rand(2,4)*5);
>> M
M =

   3   1   4   0
   3   0   3   4

>> function m = minimum(Mat)
       m = min(min(Mat))
   endfunction
>> disp("le min de la matrice : "); disp(minimum(M))
le min de la matrice :
m = 0
```

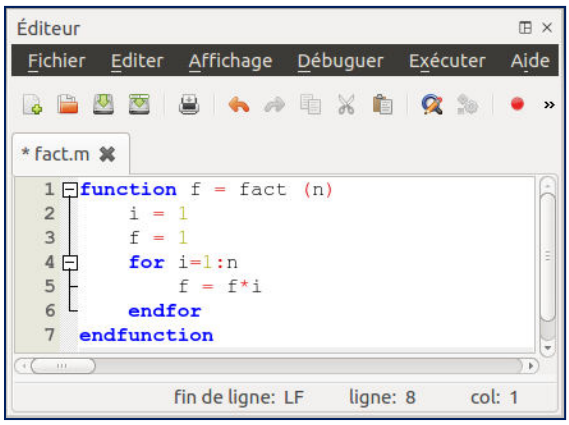
Dans l'exemple ci-dessus, la fonction a comme paramètre d'entrée une matrice « Mat » et renvoi un scalaire « m ».

4 – Scripts

Un fichier de script contient des instructions qui sont lues et exécutées séquentiellement par l'interpréteur d'Octave. Ce sont obligatoirement des fichiers au format texte. Ils sont reconnaissables par leur extension « .m ».

L'environnement graphique d'Octave mis à notre disposition un éditeur permettant d'éditer des scripts. Cependant, vous pouvez éditer ces scripts avec n'importe quel éditeur de texte.

A noter qu'une fonction peut aussi être sauvé dans un fichier de script. Dans l'exemple ci-dessous, j'ai crée une fonction dans un fichier « fact.m »



Attention, dans le cas ci-dessus, le nom de la fonction et le nom du fichier de script est le même.

L'avantage de mettre des fonctions dans des scripts, vous donne plus de flexibilité à vos programmes et il suffit de mettre le fichier de votre fonction dans votre dossier courant (ou accessible via le *path*) pour bénéficier de cette fonction.

Il faut noter aussi que vous pouvez exécuter un script de plusieurs manières :

Depuis l'éditeur en cliquant le bouton d'exécution, le menu d'exécution ou la touche F5
Depuis la fenêtre de commande en indiquant le nom du script et éventuellement des paramètres lorsqu'il s'agit d'une fonction
Depuis le *shell* du système d'exploitation.

5 – Interfaces graphiques

QCM 4 – Ecrire des programmes

1 – Entrée / sorties

Q1 – Pour afficher du texte ou le contenu de variables, Octave offre les commandes suivantes :

- ☐ write
- ☐ scanf
- ☐ print
- ☐ printf
- ☐ disp
- ☐ display
- ☐ input
- ☐ read

Q2 – Pour lire des données depuis le clavier, Octave offre les commandes suivantes :

- ☐ write
- ☐ scanf
- ☐ print
- ☐ printf
- ☐ disp
- ☐ display
- ☐ input
- ☐ read

Q3 – avec la commande « disp », Octave reconnait le type des données et les affiche correctement. Par exemple si la variable est une matrice 2x2, il met en forme correctement l'affichage

- ☐ Vrai
- ☐ Faux

Q4 – Dans la portion de code ci-dessous, j'ai caché intentionnellement le format d'affichage à donner à la commande « ***printf*** ». A vous de le trouver !

```
Fenêtre de commandes
>> notes = [12 12 9 15 16];
>> nbNotes = length(Notes);
>> moyenne = sum(notes)/nbNotes;
>> nomEtudiant = "Zidane ";
>> message = "a obtenu une moyenne de : ";
>> printf("          ", nomEtudiant, message, moyenne)
Zidane a obtenu une moyenne de :  12.80
>>
```

.....

Q5 – Je souhaite afficher la variable « salaire » sur 8 positions dont 2 décimales. Je procède comme suit :

- ☐ printf(salaire)
- ☐ printf("%", salaire)
- ☐ printf("%s", salaire)
- ☐ printf("%f8.2", salaire)
- ☐ printf("%8.2f", salaire)
- ☐ printf("%2.8f", salaire)

Q6 – Je souhaite afficher un le message "*Voici mon salaire :* " suivi d'un retour à la ligne, puis le contenu de la variable « *salaire* » sur 8 positions dont 2 décimales puis un second retour à la ligne. Je procède comme suit :

- ☐ printf(salaire)
- ☐ printf("Voici mon salaire:\n %8.2f\n", salaire)
- ☐ printf("Voici mon salaire: %8.2d ", salaire)
- ☐ printf("Voici mon salaire:\n %2.8f\n", salaire)

Q7 – Je souhaite lire à partir du clavier un vecteur « v », Je procède comme suit :

- ☐ v = input()
- ☐ v = input("donnez v : ")
- ☐ v = read("donnez v : ")
- ☐ v = scanf()

Q8 – Je souhaite lire à partir du clavier le nom d'un étudiant dans une variable « nom », Je procède comme suit :

- ☐ nom = input("donnez votre nom : ", "s")
- ☐ nom = read("donnez votre nom : ", "s")
- ☐ nom = input("donnez votre nom : ")
- ☐ nom = input("donnez votre nom : ", s)

Q9 – La commande « input » permet de lire depuis le clavier des scalaires, des vecteurs, des matrices et même des chaines de caractères.

- ☐ Vrai
- ☐ Faux

Q10 – La commande permettant de bloquer un programme jusqu'à ce que l'utilisateur tape sur une touche est :

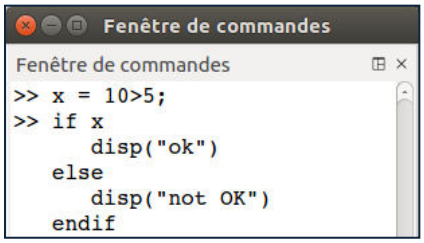
- ☐ wait
- ☐ pause
- ☐ readln()
- ☐ break

Q11 – Je souhaite bloquer un programme et le mettre en situation d'attente pendant 1 minute, je procède comme suit :

- ☐ wait (1)
- ☐ pause
- ☐ pause(1)
- ☐ pause(60)

2 – Structures de contrôle

Q12 – Que va afficher le code suivant :



- ☐ ok
- ☐ erreur
- ☐ not OK

Q13 – Que va afficher le code suivant :

```
Fenêtre de commandes
>> v = input("donnez une valeur : ");
donnez une valeur : 14
>> if isscalar(v)
      disp("1")
   else
      disp("2")
   end
```

.....

Q14 – Que va afficher le code suivant :

```
Fenêtre de commandes
>> v = input("donnez une valeur : ");
donnez une valeur : [12 15 14]
>> if ismatrix(v)
      disp("1")
   elseif isvector(v)
      disp("2")
   else
      disp("3")
   end
```

.....

Q15 – Que va afficher le code suivant :

```
Fenêtre de commandes
>> v = input("donnez une valeur : ");
donnez une valeur : [12 14; 15 16]
>> if isvector(v)
      disp("1")
   elseif isscalar(v)
      disp("2")
   elseif iscomplex(v)
      disp("3")
   else
      disp("4")
   end
```

.....

Q16 – Que va afficher le code suivant :

```
Fenêtre de commandes
>> v = input("donnez une valeur : ");
donnez une valeur : [12+5i 14+12i; 15 14]
>> if ismatrix(v) & iscomplex(v)
      disp("1")
   elseif ismatrix(v)
      disp("2")
   else
      disp(3)
   end
```

.....

Q17 – Que va contenir R à l'issue de l'exécution du code suivant :

```
Fenêtre de commandes
>> Q=5;
>> T=2;
>> if (Q>T || Q>8) & (T<=4)
       R=Q*T;
end
>> if (T==0 || Q==2 || Q>T) & (T>-5)
       R=6;
end
```

.....

Q18 – Pour lier deux conditions par un « OU » logique j'utilise l'opérateur:

☐ & ☐ || ☐ XOR ☐ OR

Q19 – Pour lier deux conditions par un « ET » logique j'utilise l'opérateur:

☐ & ☐ || ☐ XOR ☐ OR

Q20 – Que va afficher le code suivant :

```
Fenêtre de commandes
>> V = ["a","b","c"]
V = abc
>> for e =V
      disp(e)
   end
```

- ☐ les lettres « a » , « b » et « c » sur la même ligne
- ☐ les lettres « a » , « b » et « c » chacune sur une ligne à part
- ☐ une erreur

Q21 – Que va afficher le code suivant :

```
Fenêtre de commandes
>> V = [4 5 6 7 44 12 13 15];
>> ind = 1:2:length(V);
>> for i = ind
       printf("%3.0f", V(i));
   end
```

- ☐ 4 5 6 7 44 12 13 15
- ☐ 4 6 44 13
- ☐ Une erreur

TP 4 – Ecrire des programmes

Exercice 1 :

Soient les vecteurs-colonnes et la matrice suivants

$$\vec{u}_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \ \vec{u}_2 = \begin{pmatrix} -5 \\ 2 \\ 1 \end{pmatrix}, \ \vec{u}_3 = \begin{pmatrix} -1 \\ -3 \\ 7 \end{pmatrix}$$

$$A = \begin{pmatrix} 2 & 3 & 4 \\ 7 & 6 & 5 \\ 2 & 8 & 7 \end{pmatrix}.$$

A - Structures Octave

1. Entrer ces données sous Octave.
2. Calculer $\vec{u}_1 + 3\vec{u}_2 - \vec{u}_3/5$
3. Calculer le produit scalaire entre les vecteurs $\vec{u}_1$et $\vec{u}_2$
4. Calculer le produit $A\,\vec{u}_1$

B – Commandes Octave : Trouvez les commandes octave permettant de :

1. calculer les normes (mot clé **norm**) des vecteurs $\vec{u}_1$, $\vec{u}_2$ et $\vec{u}_3$
2. déterminer les dimensions de la matrice A et d'en extraire le nombre de colonnes
3. calculer le déterminant et l'inverse de *A*

C - Résolution de systèmes d'équations linéaires.

Je vous informe que vous pouvez à l'aide de la fonction « ***inv***() » calculer l'inverse d'une matrice. Vous pouvez aussi faire la division d'une matrice par un vecteur avec l'opérateur « **** ». A l'aide de cette fonction et de cet opérateur, je vous demande de résoudre le système d'équation suivant : $A\vec{x} = \vec{u}_1$.

Exercice 2 :

Je mets à votre disposition un fichier texte nommé « **notes.data** ». Ce fichier contient 20 lignes et 4 colonnes. Chaque ligne correspond à un étudiant. Chaque colonne correspond à un module (algèbre, analyse, algorithmique et programmation). Je vous demande :

1. lire depuis le fichier « notes.data » les notes dans une variable nommée « notes ». Vous utiliserez la fonction « csvread » (reportez-vous à l'aide d'octave pour avoir des explications de son usage).
2. Ecrire une fonction « eliminerNotesAberrantes.m » qui recherche dans la matrice des notes (les lignes représentent les étudiants et les colonnes représentent les modules) les lignes comportant des notes aberrantes (non comprises entre 0 et 20). Cette fonction doit rendre une matrice débarrassée de ces lignes
3. Ecrire une fonction « ***eliminerNotesManquantes.m*** » qui recherche dans la matrice des notes les lignes comportant des notes manquantes (une note manquante correspondant à la valeur NA pour dire note avalable). Cette fonction doit rendre une matrice débarrassée de ces lignes
4. Une fois les notes débarrassées des lignes comportant des notes aberrantes ou manquantes, vous devez écrire une fonction qui renvoi la moyenne pour chaque étudiant et la moyenne de la classe. Il faut aussi qu'elle affiche un graphique du taux de réussite (pourcentage des étudiants ayant eu plus de 10/20).
5. Ecrire un script « gestNotes.m » qui automatise les 4 opérations précédentes

Exercice 3 :

Question 1 : Créez un tableau ***tab*** contenant des entiers multiples de 3 compris entre 3 et 39 puis écrivez un script qui parcourt ce tableau et remplace chacune des valeurs par son carré.

Question 2 : Ecrivez une fonction « **cos2** » permettant de calculer le carré du cosinus de x

$$\cos 2 : \begin{cases} \mathbb{R} & \rightarrow & \mathbb{R} \\ x & \mapsto & cos^2(x) \end{cases}$$

Exercice 4 : Calcul des polynômes de Legendre

Les polynômes de Legendre $P_n(x)$ peuvent se calculer par les relations de récurrence suivantes :

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_n(x) &= \left(\frac{2n-1}{n}\right) x P_{n-1}(x) - \left(\frac{n-1}{n}\right) P_{n-2}(x) \end{aligned}$$

Où x est compris entre -1 et 1.

Je vous demande :

1. d'écrire une fonction permettant de calculer des polynômes de Legendre. Comme paramètre d'entrée, vous devez lui fournir les valeurs de ***n*** et ***x*** et en retour, elle doit nous fournir la valeur de $P_n(x)$. Attention, il faudrait vérifier les conditions d'applicabilité de ce calcul (x compris en -1 et 1).
2. D'écrire un script qui affiche les courbes des polynômes de Legendre $P_0(x)$, $P_1(x)$, $P_2(x)$ et $P_3(x)$ pour 100 valeurs de x (allant de -1 à 1).

Chapitre 5 – Générer des graphiques

A la fin de ce chapitre, vous serez capables :

- Gérer des fenêtres graphiques
- De générer des graphiques en 2D (*fplot*, *plot* et *logplot*)
- Rendre lisible une figure (gérer les légendes, les axes, les annotations, les étiquettes, les *subplots* et sauvegarder des figures
- Générer des graphiques en 3D (tracer de lignes de niveau d'une fonction à 2 variables, représenter une surface)

1 – Gérer des fenêtres graphiques

Octave met à notre disposition plusieurs commandes permettant d'ouvrir des fenêtres et d'afficher dedans des graphiques (courbes en 2D ou 3D, des histogrammes et bien d'autres graphiques encore que l'on peut rencontrer en statistiques par exemple).

Par défaut, l'exécution d'une commande permettant d'afficher un élément graphique va opérer sur la fenêtre courante en écrasant la figure qui est affichée précédemment.

Créer une figure :

Afin d'afficher un graphique dans une nouvelle fenêtre, il suffit d'utiliser la commande « **figure**(***n***) ». Le paramètre ***n*** indique un numéro de la fenêtre. S'il n'est pas indiqué, Octave lui affecte automatique un numéro calculé à partir du numéro de la dernière fenêtre qu'il a crée. La première fois, il lui affecte le numéro « 1 ».

Notez bien que la commande « ***figure (n)***» crée une fenêtre si elle n'existe pas déjà. Mais si elle existe, cette commande consiste à définir la fenêtre de numéro ***n*** comme fenêtre courante.

La fenêtre courante est celle où les commandes d'affichage d'éléments graphiques (courbes, labels, axes, légendes, annotations, etc.) vont être opérées.

Fermer une figure

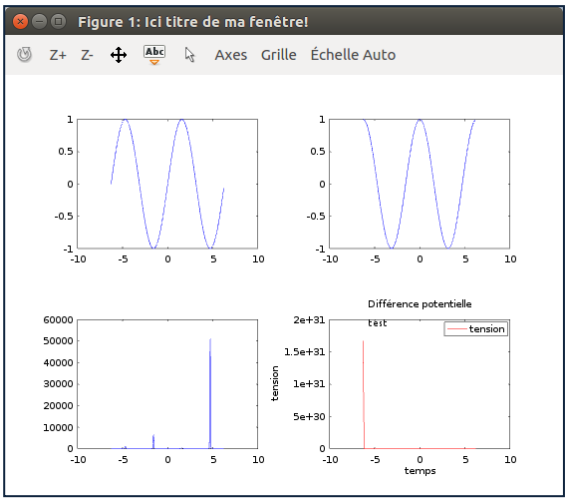
C'est la commande close qui permet de fermer une fenêtre graphique.

close	Ferme la fenêtre courante
close(i)	Ferme la fenêtre de numéro ***i***
close all	Ferme toutes les fenêtres graphiques

Constituants d'une fenêtre graphique

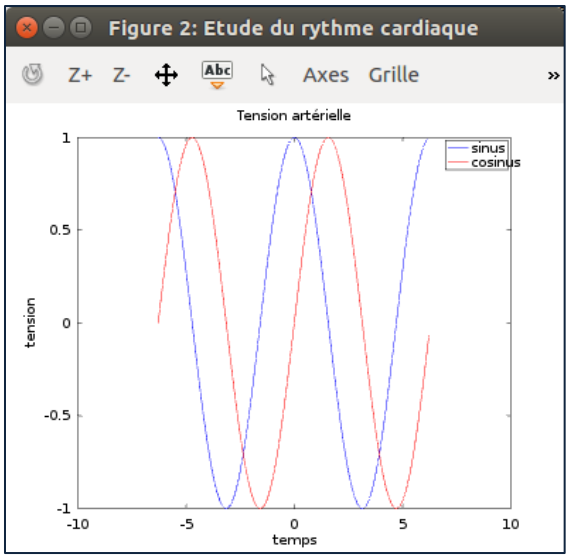
Les fenêtres graphiques Octave (ou Matlab) sont organisées de façon à répondre à la majorité des représentations graphiques exigées dans divers domaines scientifiques. Ainsi, une fenêtre est composée

- d'un titre
- d'une barre d'outils permettant d'effectuer plusieurs actions comme zoomer, annoter, afficher une grille, tourner les figures, etc.
- d'une zone d'affichage des graphiques (plots) qui peut être composé d'un ou de plusieurs graphiques
- Chaque graphique est composé de plusieurs éléments visuels :
- Axes
- Courbes
- Légende
- Etiquettes
- Titre
- Annotations
- Etc.



Exemple de figures composé de plusieurs graphiques (subplot)

Analysons une figure ci-dessous composée d'un seul graphique.



Numéro de la figure	2
Titre	Etude du rythme cardiaque
Titre du graphique	Tension artérielle
Label des abscisses (xlabel)	Temps
Label des ordonnées (ylabel)	tension
Étiquettes des abscisses	[-10, -5, 0, 5, 10]
Etiquettes des ordonnées	[-1, -0.5, 0, 0.5, 1]
Limites des abscisses	-10 à +10
Limites des ordonnées	-1 à +1
Nombre de courbes	2
Légende pour la courbe en bleu	Sinus
Légende pour la courbe en rouge	cosinus

Comme vous le voyez, dans les graphiques on retrouve des éléments visuels communs : axes, légendes, titres, étiquettes, limites des axes, etc. Dans ce qui suit, nous allons essayer de décortiquer un peut tous ça.

2 – Graphiques 2D

Au moins deux commandes Octave permettent de générer des graphiques en 2D : fplot, plot.

2.1 La commande *fplot*

Cette commande permet de tracer le graphe d'une fonction sur un intervalle donnée. Sa syntaxe est comme suit :

fplot('nomf', [x_{min}, x_{max}])

où :

- ***nomf*** est soit le nom d'une fonction Octave incorporée, soit une expression définissant une fonction de la variable x, soit le nom d'une fonction utilisateur.
- [x_{min} , x_{max}] est l'intervalle pour lequel est tracé le graphe de la fonction.

Expliquons un peu mieux tous ça avec des exemples.

Pour afficher la courbe de la fonction ***sinus*** définie sur l'intervalle [-2π, +2π] on utilisera la commande suivante :

fplot('sin',[-2*pi 2*pi])

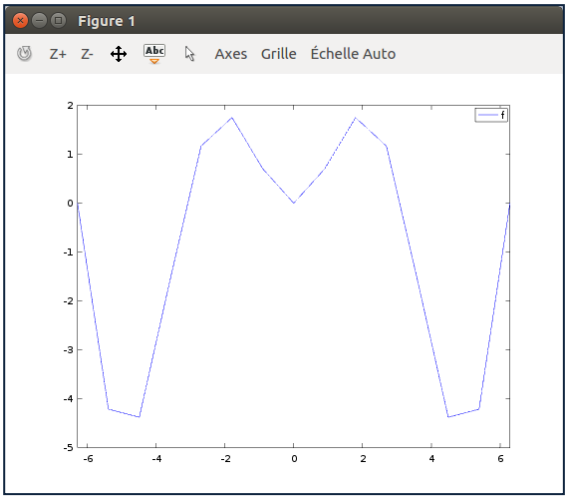
Pour tracer le graphe de la fonction « **f(x) = x sin(x) »** entre -2π, +2π, on peut définir la fonction utilisateur « **f »** dans le fichier « **f.m »** de la manière suivante (attention de bien lire x .* sin(x) et non pas x*sin(x)):

```
function y=f(x)
    y=x.*sin(x);
endfunction
```

Ainsi, il suffit d'exécuter la commande suivante pour afficher le graphe de la fonction f ci-dessus :

```
fplot('f',[-2*pi 2*pi])
```

Ce qui donne la figure ci-dessous :



Si je veux tracer plusieurs courbes dans une même figure, je procède comme suit :

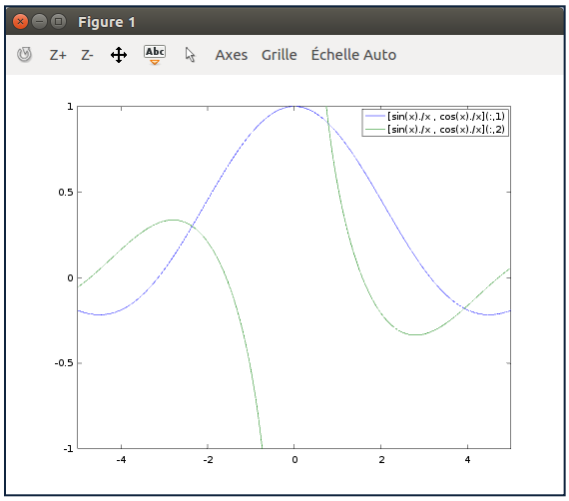
fplot('[nomf_1 , nomf_2 , nomf_3]', [xmin , xmax])

où

nomf_1, **nomf_2**, **nomf_3** est soit le nom d'une fonction octave incorporée, soit une expression définissant une fonction de la variable ***x***, soit le nom d'une fonction utilisateur.

Il est également possible de gérer les bornes des valeurs en ordonnées. Pour limiter le graphe aux ordonnées comprises entre les valeurs y_{min} et y_{max} on passera comme second argument de la commande ***fplot*** le tableau [x_{min}, x_{max}, y_{min}, y_{max}]. Une autre possibilité pour gérer les bornes des valeurs en ordonnées est d'utiliser la commande « **axis** » après utilisation de la commande **fplot**. La syntaxe est **axis([x_{min}, x_{max}, y_{min}, y_{max}])**.

Exemple :



2.2 La commande plot

A la différence de **fplot** qui se base sur l'expression d'une ou de plusieurs fonctions, la commande plot se base sur la définition d'un ensemble de points (xi , yi), i=1, ..., N. La syntaxe est :

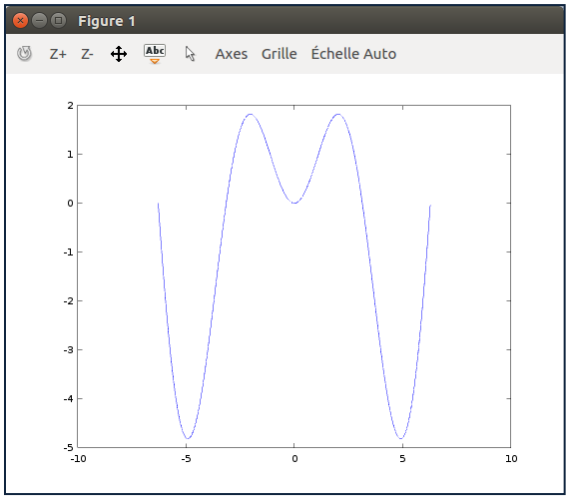
plot(x,y)

où ***x*** est le vecteur contenant les valeurs ***xi*** en abscisse et ***y*** est le vecteur contenant les valeurs ***yi*** en ordonnée. Bien entendu les vecteurs ***x*** et ***y*** doivent être de même dimension mais il peut s'agir de vecteurs lignes ou colonnes. Par défaut, les points ***(xi , yi)*** sont reliés entre eux par des segments de droites.

Exemple1 :

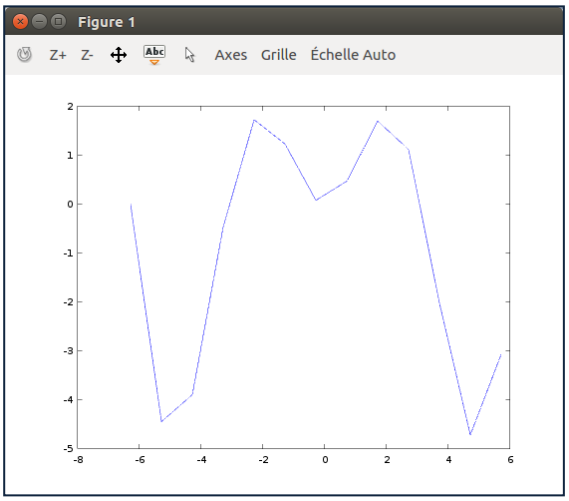
```
Fenêtre de commandes
>> x=[-2*pi:0.01:2*pi]; y = x.*sin(x);
>> plot(x,y)
```

Ce qui donne :



Exemple2 :

```
Fenêtre de commandes
>> x=[-2*pi:1:2*pi]; y = x.*sin(x);
>> plot(x,y)
```



Dans les 2 exemples ci-dessus, on a défini un vecteur ***x*** de valeurs équi-réparties entre -2π, +2π (avec un pas de 0.01 dans le premier cas et de 1 dans le deuxième cas) et on a calculé l'image par la fonction « ***xsin(x)*** » de ces valeurs (vecteur y). On affiche les points de coordonnées (x(i), y(i)).

On peut spécifier à Octave quelle doit être la couleur d'une courbe, quel doit être le style de trait et/ou quel doit être le symbole à chaque point (xi , yi). Pour cela on donne un troisième paramètre d'entrée à la commande plot qui est une chaîne de 3 caractères de la forme 'cst' avec « c » désignant la couleur du trait, « s » le symbole du point et « t » le type de trait.

Voici les couleurs que vous pouvez indiquer :

y	Jaune
m	Magenta
c	Cyan
r	Rouge
g	Vert
b	Bleu
w	Blanc
k	Noir

Voici les symboles des points possibles :

.	Point
o	Cercle
x	Marque x
+	Plus
*	Etoile
s	Carré
d	Losange
v	Triangle (bas)
<	Triangle (gauche)
>	Triangle (droit)
p	Pentagone
h	Héxagone

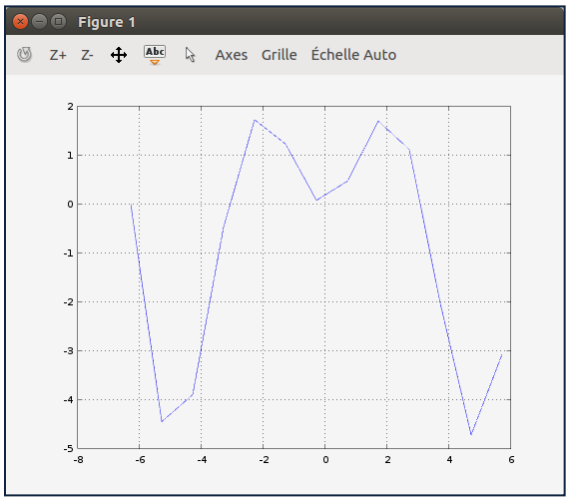
Voici les styles de traits possibles :

-	Tiret plein
:	Pointillés courts
-.	Pointillés mixte

Valeurs par défaut : **c = b, s = . et t = -** ce qui correspond à un trait plein bleu reliant les points entre eux.

Il n'est pas obligatoire de spécifier chacun des trois caractères. On peut se contenter d'en spécifier un ou deux. Les autres seront les valeurs par défaut.

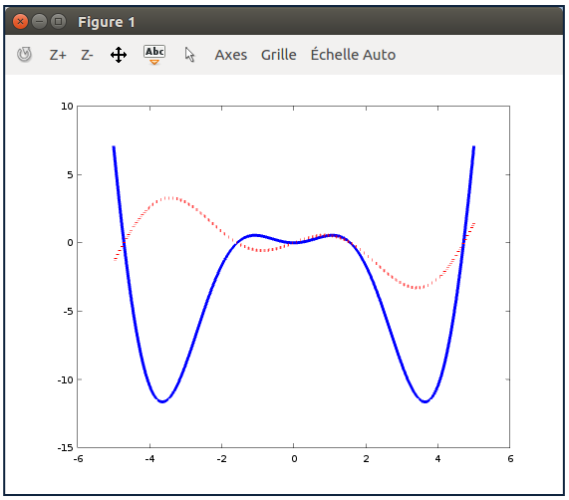
Afin de rendre plus lisible un graphique vous pouvez afficher ou cacher une grille en utilisant la commande « **grid** »



Bien évidemment, comme pour **fplot**, avec **plot**, vous pouvez afficher plusieurs graphiques dans une meme figure en spécifiant plusieurs tableaux x1, y1, x2, y2, ..., comme paramètres de la commande plot. Si l'on souhaite que les courbes aient une apparence différente, on utilisera des options de couleurs et/ou de styles de traits distincts après chaque couple de vecteurs x, y.

Exemple :

```
Fenêtre de commandes
>> x = [-5:0.01:5];
>> y = x.^2.*cos(x); z = x.*cos(x);
>> plot(x,y,'b-','linewidth', 3,
        x,z, 'r:','linewidth', 3);
```



Humm !, dans l'exemple ci-dessus, j'ai même modifié l'épaisseur du trait des cours en utilisant le paramètre « ***linewidth*** » en indiquant une épaisseur de 3!

3 - Améliorer la lisibilité d'une figure

Plusieurs éléments visuels peuvent être utilisés pour améliorer l'affichage d'un graphique. On pourra en particulier personnaliser :

- Les légendes
- Le titre
- Texte
- Etc.

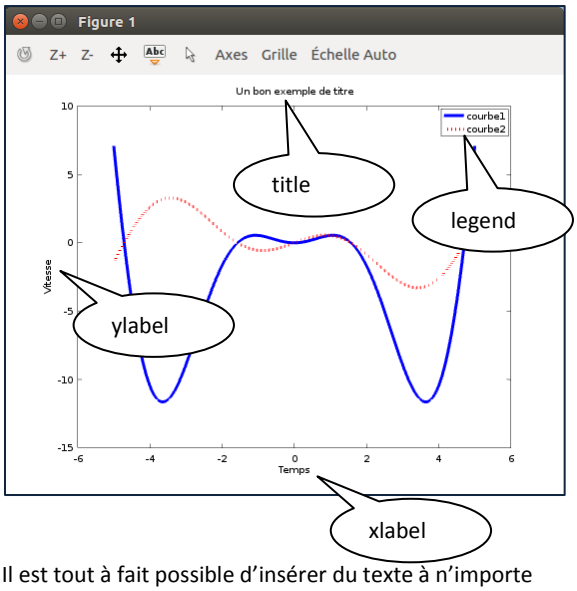
Légende :

« xlabel », « ylabel » et « legend » permettent respectivement de définir une légende pour l'axe des abscisses, l'axe des ordonnées et les courbes.

« title » permet de définir un titre à votre graphique.

Exemple :

```
Fenêtre de commandes
>> x = [-5:0.01:5];
>> y = x.^2.*cos(x); z = x.*cos(x);
>> plot(x,y,'b-','linewidth', 3,
        x,z, 'r:','linewidth', 3);
>> xlabel("Temps")
>> ylabel("Vitesse")
>> legend("courbe1", "courbe2")
>> title("Un bon exemple de titre")
```



Il est tout à fait possible d'insérer du texte à n'importe quel endroit de la figure en précisant la position exacte où doit être insérer ce texte. C'est la commande « **text** » qui permet cela. Sa syntaxe est comme suit :

Text(posx, posy, 'un texte')

Où ***posx*** et ***posy*** sont les coordonnées du point où doit débuter l'écriture du texte « un texte ».

La commande « gtext » permet de placer le texte à une position choisie sur la figure à l'aide de la souris. La syntaxe est

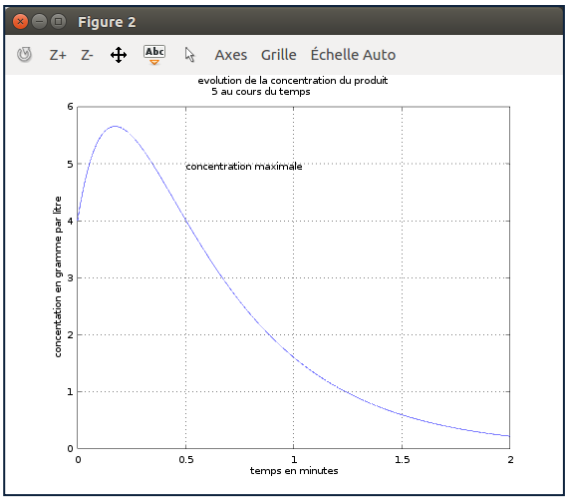
gtext(' un texte ').

Une mire, que l'on déplace en utilisant la souris, apparaît. Il suffit d'un << clic-souris >> pour que le texte apparaisse à la position sélectionnée.

Il est possible avec ces commandes d'afficher une valeur contenue dans une variable au milieu de texte. Pour cela on construit un tableau de type chaîne de caractères en convertissant la valeur contenue dans la variable en une chaîne de caractères grâce à la commande num2str. Par exemple, supposons que la variable numex contienne le numéro de l'exemple traité, disons 5. On obtiendra pour titre de la figure Exemple numero 5 par l'instruction: title(['Exemple numero ', num2str(numex)]).

Exemple :

```
Répertoire courant :
Fenêtre de commandes
>> P=5;
>> t = [0:.01:2];
>> c = 12*exp(-2*t) - 8*exp(-6*t);
>> plot(t,c); grid
>> xlabel('temps en minutes')
>> ylabel('concentation en gramme par litre');
>> title(['evolution de la concentration du produit ',
          num2str(P), ' au cours du temps ']);
>> gtext('concentration maximale');
```

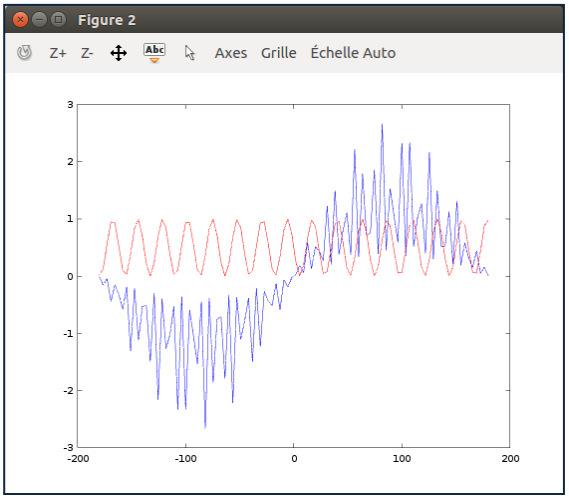


Plusieurs courbes dans une fenêtre

Il est possible d'afficher plusieurs courbes dans une même fenêtre graphique grâce à la commande ***hold on***. Les résultats de toutes les instructions graphiques exécutées après appel à la commande ***hold*** on seront superposés sur la fenêtre graphique active. Pour rétablir la situation antérieure (le résultat d'une nouvelle instruction graphique remplace dans la fenêtre graphique le dessin précédent) on tapera ***hold off***.

Exemple :

```
Fenêtre de commandes
>> x = linspace(-180,180, 100);
>> y = sind(x).*exp(cos(x));
>> z = cos(x).*sin(x)+0.5;
>> plot(x,y);
>> hold on
>> plot(x,z, 'r');
```



On dispose donc de deux façons de superposer plusieurs courbes sur une même figure. On peut soit donner plusieurs couples de vecteurs abscisses/ordonnées comme argument de la commande plot, soit avoir recours à la commande ***hold on***. Suivant le contexte on privilégiera l'une de ces solutions plutôt que l'autre.

Sous-fenêtres

Il est possible de décomposer une fenêtre en sous-fenêtres et d'afficher une figure différente sur chacune de ces sous-fenêtres grâce à la commande subplot. La syntaxe est

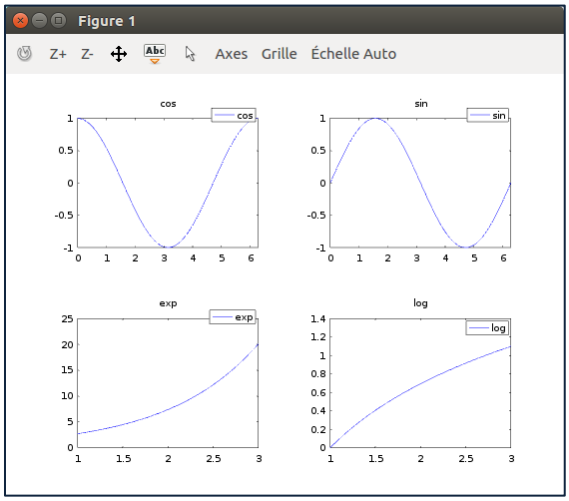
subplot(m,n,i)

où

- m est le nombre de sous-fenêtres verticalement;
- n est le nombre de sous-fenêtres horizontalement;
- i sert à spécifier dans quelle sous-fenêtre doit s'effectuer l'affichage. Les fenêtres sont numérotées de gauche à droite et de haut en bas.

Exemple :

```
Fenêtre de commandes
>> figure
>> subplot(2,2,1), fplot('cos', [0 2*pi]), title('cos')
>> subplot(2,2,2), fplot('sin', [0 2*pi]), title('sin')
>> subplot(2,2,3), fplot('exp', [1 3]), title('exp')
>> subplot(2,2,4), fplot('log', [1 3]), title('log')
```



Sauvegarder une figure

La commande print permet de sauvegarder la figure d'une fenêtre graphique dans un fichier sous divers formats d'images. La syntaxe de la commande print est:

print –f <num> -d <format> <nomfic>

où

- <num> désigne le numéro de la fenêtre graphique. Si ce paramètre n'est pas spécifié, c'est la fenêtre active qui est prise en compte.
- <nomfic> est le nom du fichier dans lequel est sauvegardée la figure. Si aucune extension de nom n'est donnée, une extension par défaut est ajoutée au nom du fichier en fonction du format choisi (.ps pour du PostScript, .jpg pour du jpeg, par exemple).
- <format> est le format de sauvegarde de la figure. Ces formats sont nombreux. On pourra obtenir la liste complète en tapant help plot. Les principaux sont:
  - ps : PostScript noir et blanc
  - psc : PostScript couleur
  - eps : PostScript Encapsulé noir et blanc
  - epsc : PostScript Encapsulé couleur
  - jpeg : Format d'image JPEG
  - tiff : Format d'image TIFF

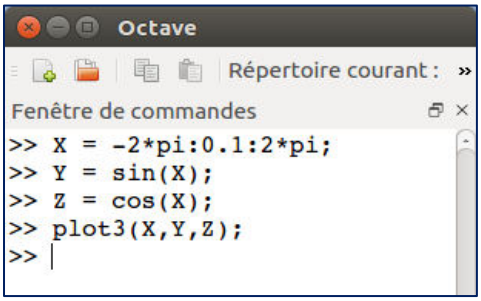
4 – graphisme 3D

4.1 – Afficher des courbes

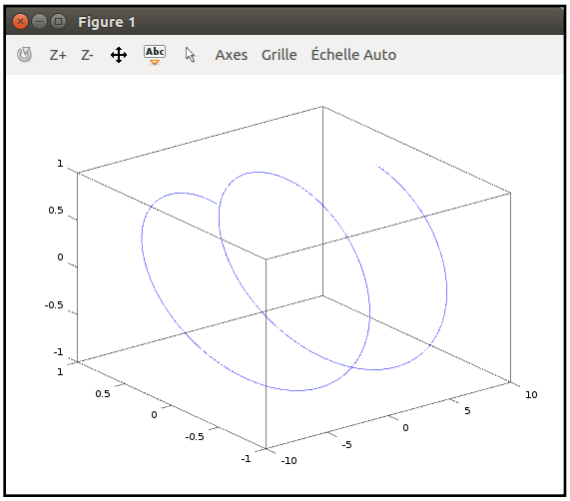
En 2D, les points sont définis par 2 coordonnées (donc 2 axes). En 3D, 3 coordonnées (x, y et z) sont nécessaires pour définir des points dans l'espace. Pour dessiner des courbes dans l'espace, il faut trois vecteurs représentant les 3 axes (X, Y et Z).

On peut, bien évidemment définir les points dans l'espace en énumérant chacune des composantes de ces points dans 3 vecteurs ayant le meme nombre d'éléments. Mais en général, il est préférable (et plus simple) de représenter ces point en définissant un vecteur (en général celui des abscisse X), puis de définir les deux autres coordonnées par une fonction. Voici une illustration :

1. je définir le vecteur des abscisses à l'aide d'une série de valeurs uniformément réparties entre -2π et 2π:
   $X = -2*\pi : 0.1 : 2*\pi$
2. puis je définie une fonction qui me permet de trouver les ordonnées à partir des abscisses. Par exemple un sinus :
   $Y = \sin(X)$.
   Ici chaque ordonnée est le sin de l'abscisse qui lui correspond.
3. je définie ensuite, une fonction qui me permet de trouver les troisièmes coordonnées (z) à partir des abscisses. Par exemple un cosinus :
   $Z = \text{con}(X)$.
4. Il suffit, enfin d'appeler la fonction « plot3 » en lui fournissant les trois vecteurs correspondants aux coordonnées des différents points dans l'espace :

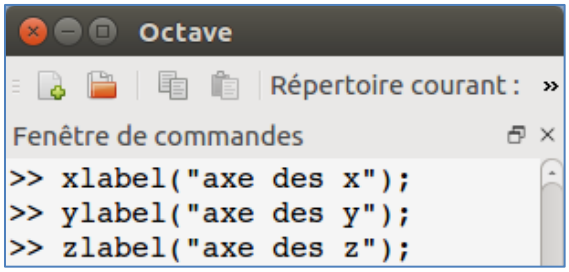


Ce qui me donne la courbe dans l'espace suivante :

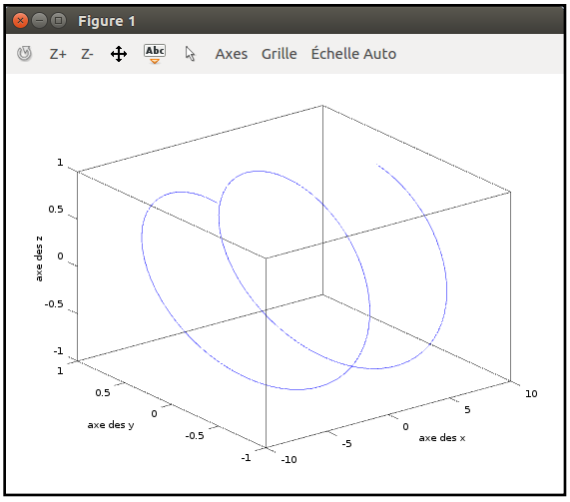


Comme en 2D, vous pourrez personnaliser comme vous le voulez cette figure en jouant sur les légendes (les axes, le titre, le légende des courbes, etc.).

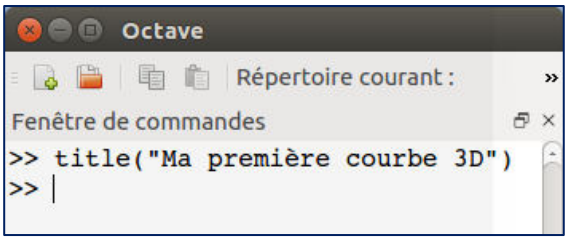
Voici un exemple où je personnalise les légendes des 3 axes :

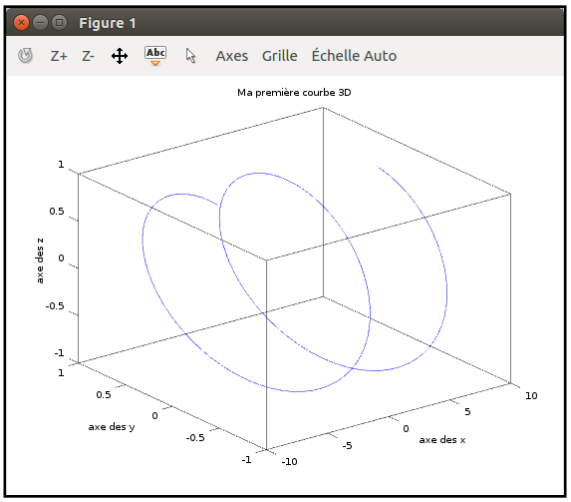


Voici ce que ça donne :



On peut aussi donner un titre à notre figure :





4.2 Courbes paramétriques

Un autre moyen de définir les coordonnées des points dans l'espace est de définir un paramètre (en lui donnant une série de valeurs), puis de définir les 3 coordonnées de nos points par des fonctions de ce paramètre. En général, ce paramètre peut être le temps ou un angle ou autre en physique par exemple.

Prenons un exemple : Supposons que nous avons un paramètre représentant un angle. Définissons plusieurs valeurs de cet angle espacées de 0.1 entre -4π et 4π: >> angle = $-2*\pi : 0.1 : 2*\pi$

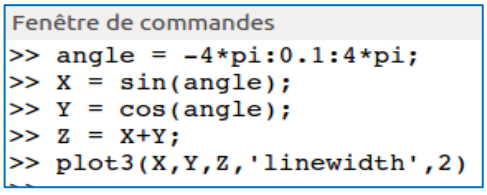
Calculons les valeurs des abscisses (les x) comme les sinus de cet angle : >> X = sin(angle)

Calculons les valeurs des ordonnées (les y) comme les cosinus de cet angle : >> Y = cos(angle)

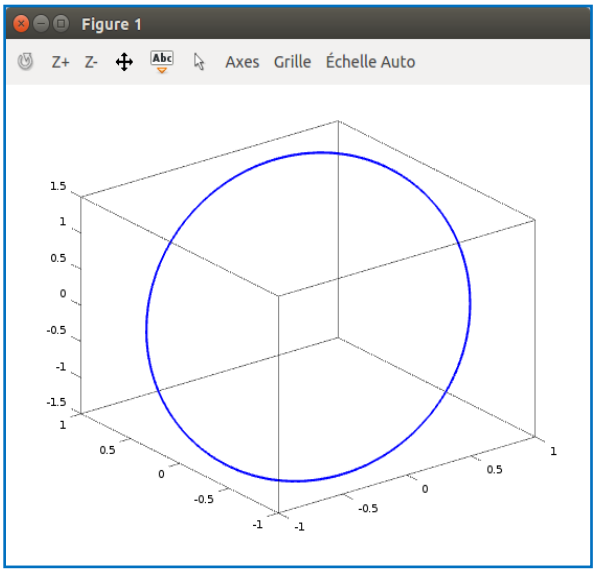
Calculons les valeurs des 3èmes coordonnées (les z) comme la somme des x et des y : >> Z = X + Y

Il ne reste plus qu'à exécuter la comme « plot3 » en luis fournissant comme paramètres les 3 vecteur X, Y et Z.

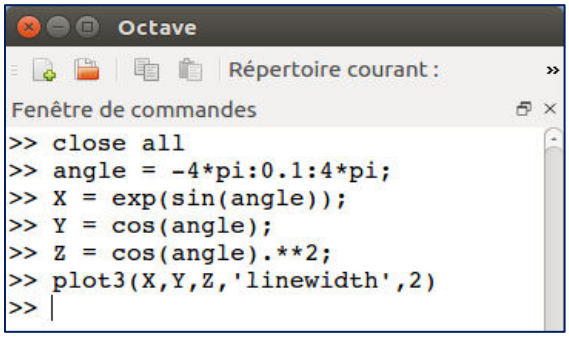
Voici ce que ça donne :

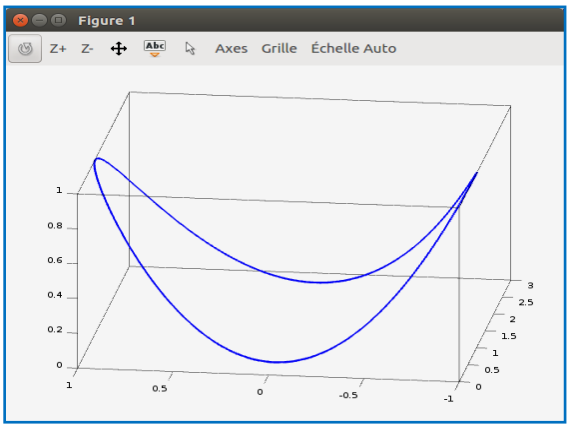


Ce qui nous donne la superbe courbe suivante :



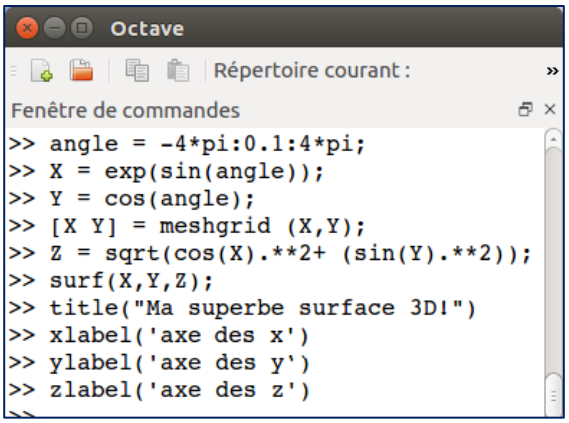
Voici un autre exemple :



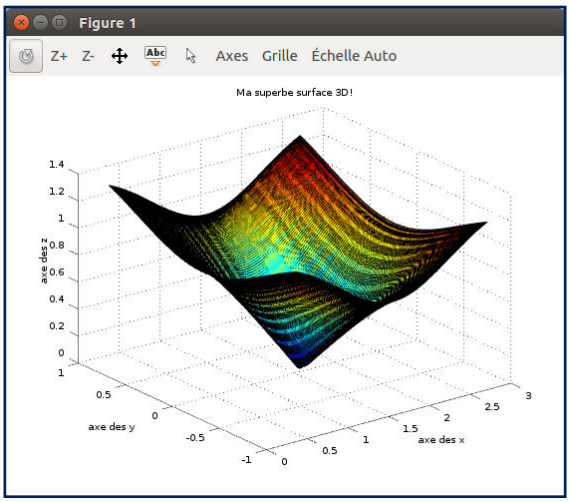


4.3 – Afficher des surfaces

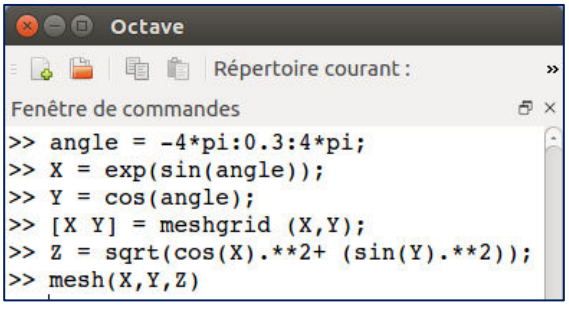
Pour générer des surfaces, il faut d'abord générer une grille de points dans un plan (en général celui des x et des y). La fonction permettant de générer cette grille est « meshgrid ». Ensuite on pourra appliquer une fonction sur les points de cette grille (plan en x et y) pour déterminer la coordonnée z. Voici un exemple :



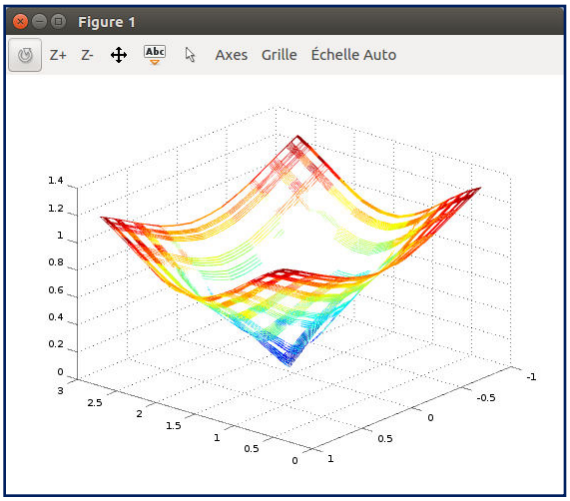
Ce qui nous donne la surface suivante :



Comme vous pouvez le constater, l'affichage d'une surface permet d'afficher une surface pleine avec des petits rectangle pleins. Il est tout à fait possible de n'afficher que les contours de ces petits rectangles, ce qui nous donne ce qu'on appelle par maillage. La fonction Octave permettant cela s'appelle « ***mesh*** ». Voici l'exemple de la surface précédente représentée par un maillage :



Voici ce que ça donne :



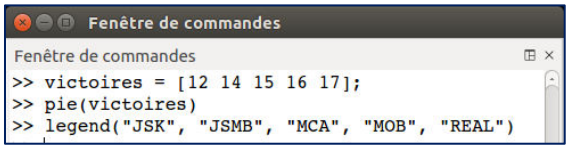
5 – Pleins de graphiques pour les statistiques

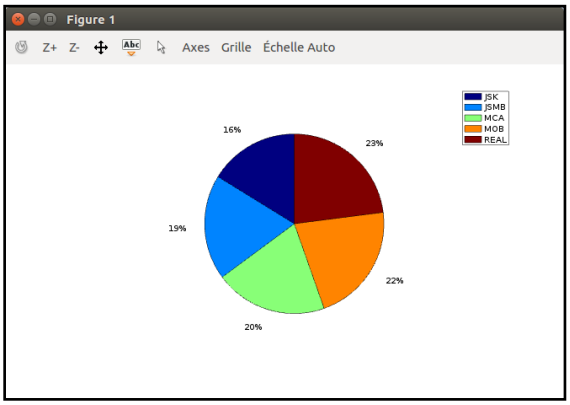
Octave permet d'afficher pratiquement tous les graphiques que vous rencontrez en statistiques : les histogrammes, les boites en fromages, les graphiques en bâtonnets, etc.

Je vous présente ici quelques exemples juste pour avoir une idée, sans rentrer dans les détails.

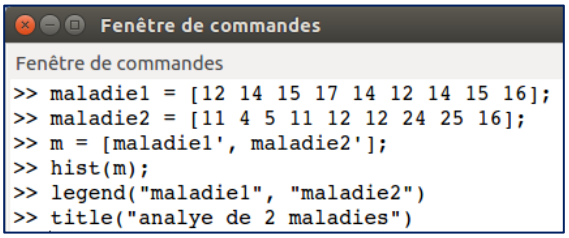
5.1 - Graphique en fromage : pie

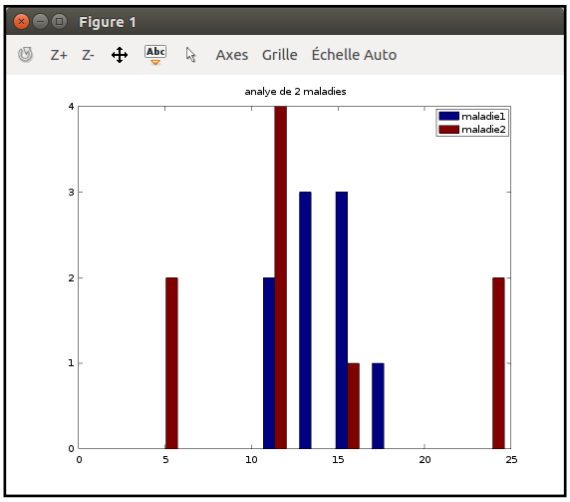
On définie une série de valeurs représentant par exemple le nombre de victoires d'une équipe de football. Puis on définie une série de légendes correspondant à cette série de valeurs. La fonction « ***pie*** » va afficher un graphique en fromage représentant la portion de chaque élément de la légende (ici des équipes de football).



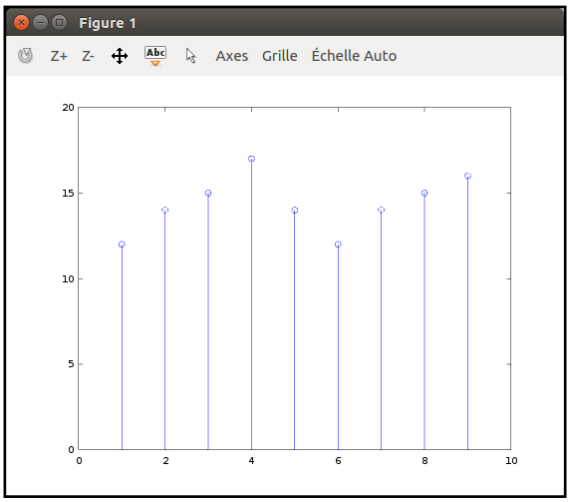


5.2 - Histogramme : hist





5.3 – Graphiques en batonnets : stem



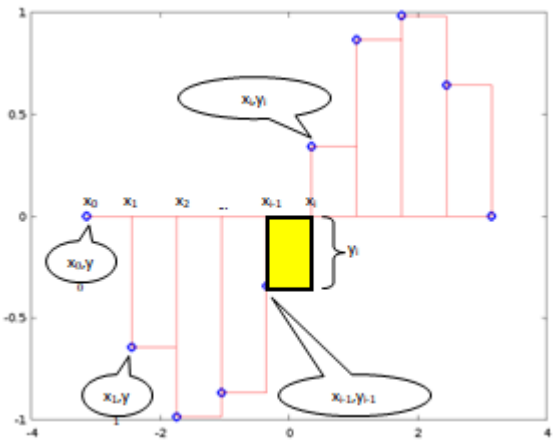
6 – exemple de problème

Afin d'illustrer l'utilité des fonctions graphiques du logiciel Octave, prenant un exemple de problème posé en analyse numérique : l'intégration numérique avec la méthode des rectangles.

<u>En entrée</u>:

- Un ensemble de points d'une courbe en 2D
  - X : vecteur des abscisses

- Y: vecteur des ordonnées
- La relation entre les ordonnées et les abscisses est définie par une fonction bien précise
- La plus petite valeur de X et sa plus grande valeur définissent l'intervalle où l'on va calculer l'intégrale



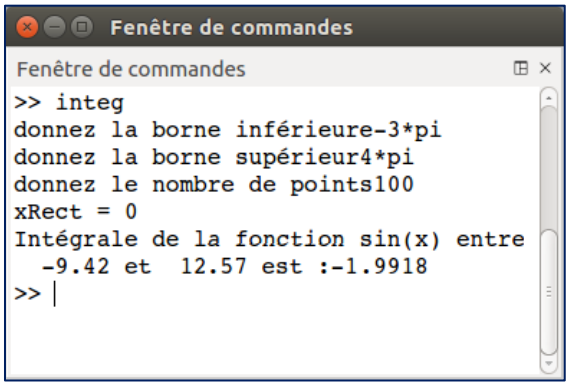
Méthode:

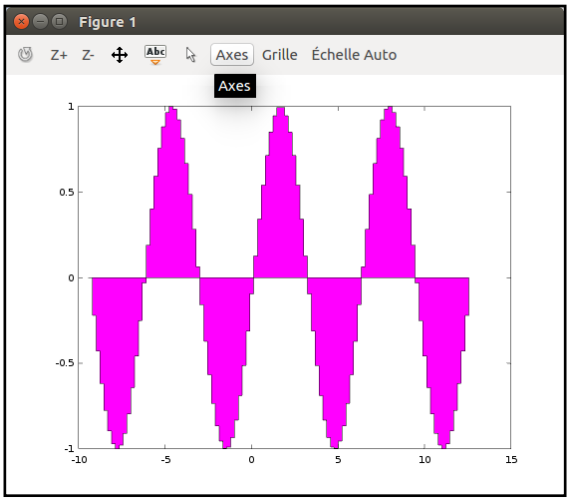
- Pour chaque couple de points successifs de la courbe, on calcul la surface du rectangle formé par ces deux points avec l'axe des abscisses.
- La somme des ces surfaces constitue l'intégrale recherchée

Voici le programme permettant de résoudre notre problème :

```
1  a = input("donnez la borne inférieure");
2  b = input("donnez la borne supérieur");
3  nbPoints = input("donnez le nombre de points");
4
5  X = linspace(a,b,nbPoints); % X va contenir les abscisses de la fonction
6  Y = sin(X); % Y va contenir les ordonnées de la fonction
7
8  res = 0.0;
9  xRect = 0.0,
10 yRect = 0.0;
11
12 for i = 2:length(X)
13   res = res + Y(i)*(X(i)-X(i-1));
14   xRect = [xRect X(i-1) X(i-1)  X(i)     X(i)     X(i-1)];
15   yRect = [yRect 0      Y(i-1)  Y(i-1)   0        0];
16 endfor
17
18 plot(X,Y);
19 plot(xRect, yRect, 'k');
20 fill(xRect, yRect, 'm');
21 printf("Intégrale de la fonction sin(x) entre %6.2f et %6.2f est :", a, b);
22 disp(res)
```

Voici ce que ça donne lors de l'exécution :





QCM 5 – Générer des graphiques

TP 5 – Générer des graphiques

Exercice 1 : Tracé d'une courbe 2D de type x y avec plot

- Définir le vecteur
  x = [0 pi/10 2pi/10 .... 2pi],
- calculer les vecteurs
  y1 = sin(x) et y2 = cos(x)
  correspondants au vecteur x,
- tracer la fonction sinus avec ***plot(x, y1)***,
- mettre un quadrillage de fond par la fonction ***grid on*** (inverse ***grid off***),
- tracer sur le même graphique la fonction y2= cos x (fonction ***hold on***, inverse ***hold off***),
- taper ***figure*** pour ouvrir une nouvelle fenêtre sans fermer la première, puis tracer
  y = exp(cos (x)).

Exercice 2 : styles de courbe

Utilisant le même vecteur x = [0 pi/10 2pi/10 ... 2pi] que dans l'exercice 1, tracer sur un même graphique les trois courbes y1 = sin(x), y2 = sin(x -0.3) et y3 = sin(x-0.5), de telle sorte que la courbe 1 soit une ligne continue rouge, la courbe 2 des cercles bleus, et la courbe 3 des pointillés noirs.

Exercice 3 : Utilisation de « subplot »

Reprendre le vecteur x= [0 pi/10 2pi/10 ..... 2pi], définir y1 = sin(x) et y2=cos(x), puis utiliser ***subplot(2,1,1)*** et ***subplot(2,1,2)*** pour tracer sur une même figure les deux graphes des fonctions sinus et cosinus, l'un en dessous de l'autre.

Exercice 4 : Options : titre, légendes …

Options du graphe : titre, labels, axes

Mot clé	Fonction
title	Définir le titre du graphe
xlabel	Label de l'axe des x
ylabel	Label de l'axe des y
zlabel	Label de l'axe des z
legend	Ajouter une légende sur le graphe
text	Permet d'ajouter du texte sur le graphe

- Tracer y = sin(x), mettre « Temps » sur l'axe des x, et « Signal » sur l'axe des y.
- Ajouter le titre : « Tension en Volts ».

Exercice 5 : **Graphiques 3D**

Dans le cours, nous avons vu trois moyens d'afficher des graphiques 3D : des courbes, des surfaces et des maillages. Sachez qu'il en existe, bien évidemment d'autres moyens.

Je vous demande d'expérimenter par vous-même ces possibilités :

- tracer la ligne paramétrique x=cos t, y = sin t, $z = t^2$ en utilisant ***plot3*** ,
  avec t=[0 pi/10,... 10pi].

Exercice 6 : **Scripts + Graphiques 2D et 3D**

Ecrivez un script qui affiche un menu composé de 3 options :

1 – affichez la courbe 2D d'une fonction

2 – afficher la courbe 3D d'une fonction

3 – Quitter

Lorsque l'utilisateur choisi l'option 1 : votre script doit faire appel à la fonction « courbe2D.m »

Lorsque l'utilisateur choisi l'option 2 : votre script doit faire appel à la fonction « courbe3D.m »

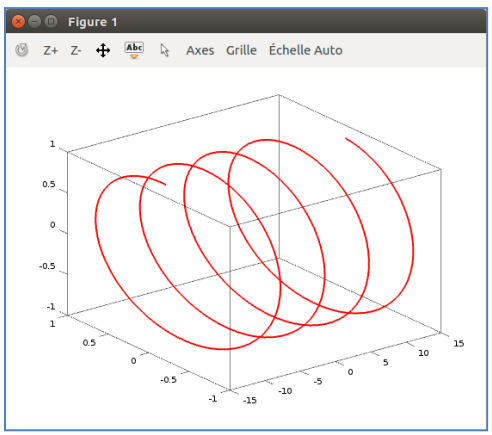
Lorsque l'utilisateur choisi l'option 3 : votre script doit s'arrêter sinon, il doit réafficher le menu.

La fonction « courbe2D.m » doit afficher le graphique 2D de la fonction $f(x)=\sin^2(x)$ sur l'intervalle $[-2\pi, +2\pi]$.

Vous devez afficher le titre du graphique qui est « cours 2D » le label de l'axe x est « temps », le label de l'axe des y est « tension », la couleur de la courbe doit être rouge, l'épaisseur du trait doit être 2.

La fonction « courbe3D.m » doit afficher le graphique 3D de la courbe définie par les vecteurs x, y et z.

- le vecteur x est composé de 100 valeurs équidistantes comprises entre $[-4\pi, +4\pi]$
- les éléments de y sont les sinus des éléments de x
- les éléments de y sont les cosinus des éléments de x



Chapitre 6 – Calcul algébrique

6.1 – Calcul algébrique sur les matrices

Octave, Matlab, Scilab, Python (pylab) et bien d'autres logiciels offrent des fonction pratiques et très utiles de manipulation algébriques de fonctions et de structures comme des vecteurs ou des matrices.

Voyons une partie de ce que permet Octave...

1 - Résoudre des équations non-linéaires

La commande "fsolve("f",x0)" permet de donner une approximation de la solution à l'équation f (x) = 0 en partant du nombre initiale "x0".

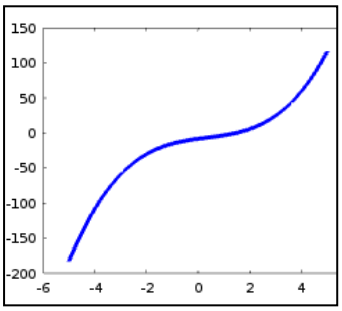
Par exemple si on veut résoudre: $x^3-x^2+5x-8=0$

Etape 1 : Définsissons d'abord notre fonction *f*:

```
function y = f(x)
    y = x.^3 - x.^2 + 5.*x - 8;
endfunction
```

Etape 2 : **Dessinons** le graphique de la fonction f pour voir approximativement ou se situe le zero de la fonction:

```
x = -5:0.1:5;
plot(x, f(x))
```



Etape 3 : **Identifions** à peu près ou se trouve le zéros de la fonction f en observant le graphique.

- On voit que le zéro se trouve entre 0 et 2 donc on peut choisir x_0=1.5x0=1.5:

Etape 4 : **Résolvant** l'équation f(x)=0 en indiquant une valeur initiale de recherche de 1.5 pour la fonction fsolve :

```
z = fsolve("f",1.5)
z =  1.4265
```

Etape 5 : Youpy, le résultat est 1.4265 : On peut donc écrire f(1.4265) = 0

Vérifions cela: y = f(z)

2 - Résoudre un Système d'équations linéaires

Illustrons le processus de résolution des système d'équation linéaire par l'exemple suivant:

$$\begin{cases} 2x_1 + 3x_2 - x_3 = -1 \\ x_1 - x_2 + 3x_3 = 4 \\ 2x_1 - 3x_2 + x_3 = 3 \end{cases}$$

Bien évidemment, on peut écrire ce système d'équation sous forme matricielle comme suit: ***A.x = b***.

Etape 1 : Définir le vecteur :

$$b = \begin{pmatrix} -1 \\ 4 \\ 3 \end{pmatrix}$$

```
b = [-1 ; 4 ; 3]
b =

   -1
    4
    3
```

Etape 2 : Définir la matrice

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 1 & -1 & 3 \\ 2 & -3 & 1 \end{pmatrix}$$

```
A = [2 3 -1; 1 -1  3; 2 -3 1]
A =

   2   3  -1
   1  -1   3
   2  -3   1
```

Etape 3 : Poser le système d'équations avec le vecteur d'inconnues x : Ax = b

$$\begin{pmatrix} 2 & 3 & -1 \\ 1 & -1 & 3 \\ 2 & -3 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -1 \\ 4 \\ 3 \end{pmatrix}$$

Ce qui nous donne : (inverse de A)(A)X =(inverse de A)B qu'on écrit, en octave, par :

x = inv(A)*b ou x = b\A

```
x = A\b
x =

   0.50000
  -0.31250
   1.06250
```

Ou

```
x = inv(A)*b
x =

   0.50000
  -0.31250
   1.06250
```

Etape 4 : Vérifions

```
A*x
ans =

  -1.00000
   4.00000
   3.00000
```

6.2 Calcul polynomial

1 – Représentation des polynmes

Octave et Matlab représentent en interne un polynôme par ses coefficients en adoptant un ordre descendant sauvés dans un vecteur.

Dans l'exemple suivant, nous représentons le polynôme:

$$p(x) = 3x^3 - 5x + 1.5$$

Ce qui peut s'écrire :

$$p(x) = 3x^3 + 0x^2 - 5x + 1.5$$

```
# Voici un exemple depolynôme
p = [3 0 -5 1.5];
degreDeP = columns(p)-1

degreDeP =  3
```

2 – Instanciation

Evaluer un polynôme avec une seule valeur

En utilisant la fonction "polyval" On peut calculer la valeur d'un polynôme pour une valeur de x donnée. Par exemple, dans ce qui suit, je calcul la valeur du polynôme précédent p1 pour x = 3:

```
polyval(p,3)  %ici je vcalcul p(x=3)
ans =  67.500
```

```
x = 1.10119;
3*x^3 + 0*x^2-5*x +1.5
ans =    2.3124e-05
```

et avec plusieurs valeurs

Il est possible d'évaluer un polynôme avec des valeurs issues d'un vecteur:

```
valeurs =[3 4 5 6 7];
polyval(p, valeurs)
ans =

    67.500   173.500   351.500   619.500   995.500
```

3 – Dérivation

Dérivée simple : La dérivée du polynôme :

$$p(x) = 3x^3 - 5x + 1.5$$

est : $p'(x) = 9x^2 - 5$

```
polyder(p)
ans =

   9   0  -5
```

Dérivée nième! Par exemple le dérivé second du polynôme p (x) est le dérivé de la dérivée de p (x):

Voici un polynôme: $P(x) = 3x^3 - 5x + 1.5$

sa première dérivée est ; $P'(x) = 9x^2 - 5$

sa seconde dérivée est ; $P''(x) = 18x$

```
printf("voici la polynôme p(x)");
disp(p);
printf("\nvoici sa dérivée");
derivePremiere =polyder(p);
disp(derivePremiere)
printf("\nvoici sa dérivée seconde ");
deriveSeconde =polyder(derivePremiere);
disp(deriveSeconde)

voici la polynôme p(x)
   3.00000   0.00000  -5.00000   1.50000

voici sa dérivée
   9   0  -5

voici sa dérivée seconde
   18    0
```

4 - Intégration

L'intégrale indéfinie d'une fonction *f* (ou d'un polynôme) est une fonction F tel que *f* est sa dérivé. En d'autres termes: F'=*f*.

Octave permet de trouver, à l'aide de la fonction **polyint** l'intégrale d'un polynôme donné.

Par exemple: $p(x) = \int p'(x)dx = \int (9x^2 - 5)\,dx$

```
printf("intégrale de la dérivé de p(x):");
disp(polyint(derivePremiere));
printf("\nPour rappel voici p(x):");
disp(p);

intégrale de la dérivé de p(x):
   3   0  -5   0

Pour rappel voici p(x):
   3.00000   0.00000  -5.00000   1.50000
```

5 – Racines d'un polynôme

Cherchons les racine de notre polynôme **p(x)**

$$p(x) = 3x^3 - 5x + 1.5$$

```
printf("Voici les racines du polynome p(x)");
disp(roots(p))

Voici les racines du polynome p(x)
  -1.42077
   1.10119
   0.31958
```

6 – Afficher correctement un polynôme

```
printf("Voici notre polynôme p(x)");
disp(polyout(p,'x'));

printf("\nVoici sa dérivée p'(x)");
disp(polyout(polyder(p),'x'));

printf("\nVoici sa dérivée seconde p''(x)");
disp(polyout(polyder(polyder(p)),'x'));
```

```
Voici notre polynôme p(x)
3*x^3 + 0*x^2 - 5*x^1 + 1.5

Voici sa dérivée p'(x)
9*x^2 + 0*x^1 - 5

Voici sa dérivée seconde p''(x)
18*x^1 + 0
```