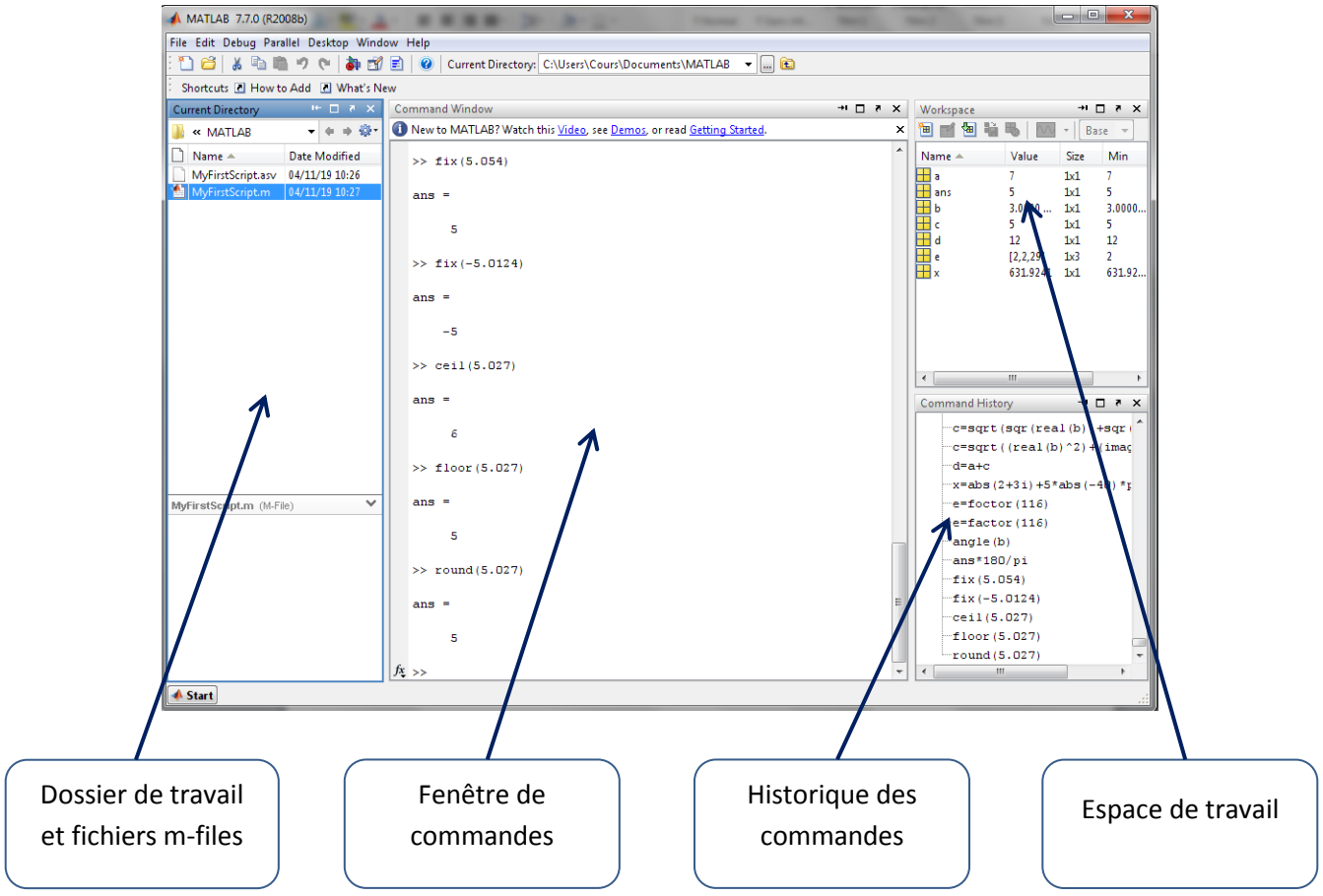
Introduction à MATLAB

I. Introduction :

MATLAB est l'abréviation de ***MATrix LABoratory***. C'est un environnement de développement dans le domaine du calcul matriciel numérique. Ce logiciel est développé par la société ***The MathWorks***.

Il permet de manipuler des tableaux et des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de tracer des graphiques en deux et en trois dimensions, de résoudre des équations et des polynômes, etc.

Interface de MATLAB :



II. Les variables sous MATLAB:

Dans **MATLAB**, les variables et les scalaires sont traités tous comme des matrices (Par exemple, un scalaire (qui est une variable d'un seul élément) serait une matrice de 1 x 1).

- La variable ***ans*** (pour dire ***answer***) contient le résultat de la dernière opération si on n'avait pas définie une variable pour contenir ceci.
- Le signe **%** est utilisé pour insérer des commentaires.
- Le signe **=** est utilisé pour l'affectation :
  - x = 35
  - x = 'bonjour'.
- MATLAB affiche le résultat après l'exécution de chaque commande sauf si celle-ci est suivie par un point-virgule « ; ».

III. Quelques Principales Commandes :

- **help** : Pour plus d'informations sur une commande

  Exemple :
  - ***help format*** : affiche une explication sur la commande format.
  - ***help help*** : afficher toutes les commandes du MATLAB arrangées par catégories.
- **quit :** Cette commande est utilisée pour quitter MATLAB, à la fin de notre travail.
- **clc :** Pour effacer la fenêtre.
- **clear :** elle efface la fenêtre des commandes.
- **clear all :** Elle réinitialise l'espace de travail (le "workspace") en détruisant toutes les variables actives en mémoire.
- **who :** afficher l'ensemble des variables actives.
- **whos :** afficher l'ensemble des variables actives en détail.
- **disp() :** afficher un texte ou bien la valeur d'une variable .

  Exemple :
  - ***disp***('c''est un message') / ***disp(a).***
- **input :** pour lire La valeur d'une variable au clavier.

  Exemple :
  - ***b=input***('Entrer la valeur de b : ') : après l'affichage du message MATLAB reste en attente d'une valeur à faire entrer au clavier.

IV. Opérations arithmétiques sur les scalaires :

- Addition : a=5+3
- Soustraction : c=b-3
- Multiplication : b=a*c
- Division : f=a/2 (division à gauche)
- Division : f = a\2 (division à droite).
- Puissance : m=5^2
- Modulo : x=mod(a,b).
- Multiplication élément par élément: c=A.*B

V. Opérations sur les vecteurs et les matrices :

- ***X1 = [1 2 3], X2 = [4,5,6,7], X3 = [8; 9; 10]*** : X1 et X2 sont deux vecteur lignes (horizontaux) respectivement de 3 et 4 éléments, et X3 est un vecteur colonne(vertical) de 3 éléments.
- ***A=[2 : 5]*** : déclare un vecteur avec les valeurs 2, 3,4 et 5.
- ***A=[2 : 0.5 : 5]*** : déclare un vecteur avec les valeurs 2, 2.5, 3, ..,5 (pas = 0.5).
- ***X = linspace(1.1,1.9,9)*** **:** crée un vecteur de 9 éléments de la suite 1.1 à 1.9 (le pas sera défini selon le nombre d'éléments).
- ***ones(1,n)*** : vecteur ligne de longueur **n** dont tous les éléments valent 1.
- ***ones(m,1)*** : vecteur colonne de longueur **m** dont tous les éléments valent 1.
- ***zeros(1,n)*** : vecteur ligne de longueur **n** dont tous les éléments valent 0.
- ***zeros(m,1)*** : vecteur colonne de longueur **m** dont tous les éléments valent 0.
- ***rand(1,n)*** : vecteur ligne de longueur **n** dont les éléments sont générés de manière aléatoire entre 0 et 1.
- ***rand(m,1)*** : vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1.
- ***prod(V)*** : produit des éléments d'un vecteur, si V est une matrice le résultat est un vecteur vertical dont chacun de ses éléments correspond au produit des éléments de chaque ligne de la matrice V.
- ***length(V)*** : donne la longueur d'une chaine de caractère, ou le nombre d'éléments d'un vecteur.

Matrices :

- ***A = [1 2 3 ;4 5 6 ;7 8 9 ;0 1 2]*** : déclare une matrice de 4 lignes et 3 colonnes.
- ***length(A)*** : donne le nombre de lignes de la matrice A.
- ***size(A)*** donne le nombre de lignes et le nombre de colonnes de la matrice A. On peut les obtenir de manière séparée le nombre de lignes et de colonnes par les instructions **size(A,1)** et **size(A,2).**
- ***c=magic(n)*** : crée un carré magique de dimensions n x n (de la suite 1 à n*n).
- ***eye(n)*** : crée la matrice identité dans $R^{n,n}$.
- ***ones(m,n)*** : crée une matrice à **m** lignes et **n** colonnes dont tous les éléments valent 1.
- ***zeros(m,n)*** : crée une matrice à **m** lignes et **n** colonnes dont tous les éléments valent 0.
- ***rand(m,n)*** : crée une matrice à **m** lignes et **n** colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1.
- ***diag(A)*** permet d'obtenir la diagonale d'une matrice.
- ***diag(A,1), diag(A,-2)*** permettent respectivement la 1ère sur-diagonale et la 2ème sous-diagonale de A (si celles-ci existent).
- ***tril(A)*** et ***triu(A)*** créent une matrice triangulaire inférieure respectivement supérieure à partir de la matrice A.
- ***tril*** et ***triu*** admettent un 2ème paramètre similaire à celui de ***diag***.
- ***det(A***) : calcule le déterminant de A.

- ***A'*** ou ***transpose(A) :*** ces deux commandes sont utilisées pour calculer la matrice transposée.
- ***inv(A) :*** calcule l'inverse d'une matrice.
- ***eig(A) :*** calcule les valeurs propres d'une matrice.
- ***syms a b c :*** déclare des variables a , b et c.

Référencement (indexation) des éléments de la matrice :

- ***A(2,3)*** : désigne l'élément de la 2ème ligne et la 3ème colonne (indexation classique).
- ***A(5)*** : désigne l'élément 5ème dans la matrice (indexation linéaire).
- ***(A>5)*** : désigne les éléments de A qui sont supérieurs à 5. Le résultat sera une matrice de mêmes dimensions que A et dont les éléments de A satisfaisant la condition seront représentés par des 1 et les autres par des 0 dans la matrice résultat. Pour obtenir les valeurs de ces élément il faut appliquer **A(A>5).**
- ***find(A>5)*** donne un vecteur des indexes linéaires des éléments de A supérieurs à 5. Pour obtenir les valeurs de ces élément il faut appliquer **A(find(A>5)).**
- Le mot clé '***end***' est utilisé pour désigner le dernier élément dans une ligne ou dans une colonne. ***A(2,end)*** et ***A(end,1)*** désignent respectivement le dernier élément de la 2ème ligne et le dernier de la 1ère colonne. ***A(end,end)*** désigne l'élément de la dernière ligne et la dernière colonne.
- ***A(1,:)*** désigne la 1ère ligne dans la matrice A et ***A(:,2)*** désigne la 2ème colonne ; ***A(:, :)*** (c'est équivalente à ***A(:)***) désigne tous les éléments de la matrice.
- ***A(:)=100*** : remplit la matrice A (déjà crée) par 100.
- ***A = repmat (100, 4, 3)*** crée une matrice de 4x3 de valeur 100.
- ***X=repmat(A,3,2)*** : copy la matrice A dans X en répétant la matrice A 2 fois horizontalement et 3 fois verticalement. (ceci est équivalent à ***X=A([1 2 1 2 1 2] ,[1 2 3 1 2 3]).***
- ***A = reshape(A,3,2)*** : redimensionne la matrice A (elle devient 3x2 tout en conservant le même nombre d'éléments).
- ***v = A(:,2); A(:,2) = A(:,3); A(:,3) = v*** : l'ensemble de ces commandes permet de permuter les deux colonnes 2 et 3 de la matrice A.
- ***L=[2,3] ; C=[1 4 5] ; X=A(L,C)*** : X sera égale à la partie de A désignée par l'intersection des lignes référencées dans L et les colonnes référencées dans C.

Concaténation de matrices et vecteurs :

- ***X=[A, V]*** : concaténation horizontale A et V doivent avoir le même nombre de lignes.
- ***X=[A ;V]*** : concaténation verticale A et V doivent avoir le même nombre de colonnes.
- Suppression d'une ligne ou d'une colonne :
- ***A(3 , :)=[]*** : suppression de la 3ème ligne de A.
- ***A(:, 2:)=[]*** : suppression de la 2ème colonne de A.

Autres :

- ***nnz(A)*** : donne le nombre d'éléments de A non nuls.

- ***B=sparse(A)*** : crée une matrice B à partir de A selon la structure **sparse** (adéquate pour les matrices creuses).
- ***spy(A)*** : affiche un graphe dont les lignes sont représentées sur l'axe des ordonnées et les colonnes sur celui des abscisses, et les points représentent les éléments non nuls.
- Le passage de l'indexation classique à l'indexation linéaire est calculé par : ***M(i,j) => M(i+(j-1)*size(M,1)).*** Ceci est possible en utilisant la fonction ***sub2ind***.

  Exemple : ***idx = sub2ind(size(M),4,3).***
- Le passage de l'indexation linéaire à l'indexation classique est calculé par : ***M(k) => M(k-ceil(k/size(M,1))*size(M,1)+size(M,1),ceil(k/size(M,1))).*** Ceci est possible en utilisant la fonction ***ind2sub***.

  Exemple : ***[ligne,colonne] = ind2sub(size(M),12).***
- **sqrt(x)** : racine carrée de x,
- **log(x) :** logarithme népérien de x,
- **exp(x) :** exponentielle de x.
- **mod(n,m) :** n modulo m, reste de la division euclidienne de m par n.
- **abs(z) :** module de z,
- **angle(z) :** argument d'un nombre complexe z.
- **real(z) :** partie réelle de z,
- **imag(z) :** partie imaginaire de z,
- **i** ou **j** : nombre complexe i = e iπ/2 , sauf si on change la valeur de i ou j dans une boucle par exemple,
- **sort** : permet de trier un vecteur (regarder l'aide), la commande sort renvoie également la permutation qui permet de passer du vecteur au vecteur ordonné. Il arrive souvent que seule cette permutation soit intéressante.
- **length(A)** : donne la plus grande dimension d'une matrice A,
- **sum(V)** : somme les composantes d'un vecteur V,
- **diff(V)** : renvoie un vecteur dont les composantes sont les différences des composantes consécutive du vecteur V,
- **while ... end** : permet de faire une boucle while,
- **a==b** : renvoie le booléen 1 si a = b et 0 si a 6= b,
- **A'** : matrice transposée de A,
- **floor(x)** : partie entière de x.
- **ceil(x)** : partie entière supérieure de x.
- **max(x)** : maximum d'un vecteur x.
- **min(x)** : minimum d'un vecteur x.
- **mean(x)** : moyenne d'un vecteur x.

VI. Instructions de contrôle :

Boucle FOR	Boucle WHILE	Le traitement conditionnel IF	Choix ventilé, l'instruction switch
for indice = borne_inf : borne_sup séquence end	while expression logique séquence end	if expression logique séquence1 else séquence2 end	switch var case cst_1 , sequence 1 case cst_2 , séquence 2 ... case cst_N , séquence d'instructions N otherwise séquence par défaut end

for r=1.1:-0.1:0.75
disp(['r = ',
num2str(r)]);
end
	while k <= n
	nfac = nfac*k;
	k = k+1;
	end
		if x<5 x=0
		else
		x=10
		end
			switch numex
			case 1,
			A = ones(n)
			case 2,
			A = magic(n);
			case {3,4},
			A = rand(n);
			otherwise
			error('numéro d'exemple non prévu ...');
			end

VII. Les programmes MATLAB (fichiers-M)

Afin d'éviter à avoir à retaper une série de commandes, il est possible de créer un programme MATLAB, connu sous le nom de «fichier-M» («M-file»), le nom provenant de la terminaison « .m » de ces fichiers.

Il s'agit, à l'aide de l'éditeur de MATLAB (Menu « File ! New ! M-File »), de créer un fichier en format texte qui contient une série de commandes MATLAB. Une fois le fichier sauvegardé (sous le nom *nomdefichier.m* par exemple), il s'agit de l'appeler dans MATLAB à l'aide de la commande:

```
>> nomdefichier
```

Les commandes qui y sont stockées seront alors exécutées. Si vous devez apporter une modification à votre série de commandes, vous n'avez qu'à modifier la ligne du fichier-M en question et à réexécuter le fichier-M en entrant le nom du fichier dans MATLAB à nouveau (essayez la touche "). Cette procédure vous évite de retaper une série de commandes à répétition. C'est la procédure recommandée pour vos travaux pratiques.

Il est possible de programmer des boucles et des branchements dans les fichiers-M:

```
for i=1:10        % boucle for
  if i<5          % branchement
    x(i)=i;       % n'oubliez pas votre point-virgule, sinon...
  else            % on peut utiliser le else
    x(i)=0;
  end             % on n'oublie pas de terminer le branchement
end               % et la boucle
```

Le point-virgule à la fin d'une ligne signale à MATLAB de ne pas retourner le résultat de l'opération à l'écran. Une pratique courante est de mettre des «;» à la fin de toutes les lignes et d'enlever certains de ceux-ci lorsque quelque chose ne tourne pas rond dans un programme, afin de voir ce qui se passe.

En général, on essaie d'éviter un tel programme. Puisque l'on utilise un langage interactif, ce bout de code fait un appel au logiciel à chaque itération de la boucle et à chaque évaluation de la condition. De plus, le vecteur change de taille à chaque itération. MATLAB doit donc faire une demande au système d'exploitation pour avoir plus de mémoire à chaque itération. Ce n'est pas très important pour un programme de petite taille, mais si on a un programme qui fait une quantité importante de calculs, un programme optimisé peut être accéléré par un facteur 1000!

Il est possible d'éviter ces goulots d'étranglement en demandant initialement l'espace mémoire nécessaire à vos calculs, puis en *vectorisant* la structure du programme:

```
x=zeros(1,10);      % un vecteur de dimension 1x10 est initialisé à zéro
x(1:4)=1:4;         % une boucle à l'aide de l'opérateur :
```

Dans cet exemple, l'espace mémoire n'est alloué qu'une seule fois, et il n'y a que deux appels faits à MATLAB.

Notons que le concept de vectorisation n'est pas particulier à MATLAB. Il est aussi présent dans les langages de programmation d'ordinateurs vectoriels et parallèles. Ceci étant dit, précisons que la vectorisation d'algorithmes est un « art » difficile qui demande du travail.

Les programmeurs expérimentés prennent l'habitude de débuter leurs programmes MATLAB avec une série de leurs commandes préférées, qui ont pour but d'éviter les mauvaises surprises. Par exemple:

```
>> clear all;           % on efface toutes les variables, fonctions, ...
>> format compact;      % supression des retours de chariots superflus
>> format short e;      % éviter qu'une petite quantitée s'affiche comme 0.0000
>> dbstop if error      % faciliter l'impression d'une variable en cas d'erreur
```

VIII. Les fonctions MATLAB

En plus des fonctionnalités de base de MATLAB, une vaste bibliothèque de fonctions (les « *toolbox* » en langage MATLAB) sont à votre disposition. Pour avoir une liste des familles de fonctions qui sont disponibles, entrez la commande help. Pour voir la liste des fonctions d'une famille de fonctions, on peut entrer help *matfun* par exemple, afin de voir la liste des fonctions matricielles. Pour obtenir de l'information sur une fonction en particulier, il s'agit d'utiliser la commande *help avec* le nom de la fonction, soit *help cond* pour avoir de l'information sur la fonction *cond*. Si la fonction n'a pas été compilée afin de gagner de la vitesse d'exécution, il est possible de voir le code source en entrant type *cond*, par exemple.

Il est aussi possible de créer ses propres fonctions MATLAB. Le concept de fonction en MATLAB est similaire aux fonctions avec d'autres langages de programmation, i.e. une fonction prend un/des argument(s) en entrée et produit un/des argument(s) en sortie. La procédure est simple. Il s'agit de créer un fichier-M, nommons-le *mafonction*.m. Ce qui différentie un fichier-M d'une fonction est que la première ligne de la fonction contient le mot clef *function*, suivi de la définition des arguments en entrée et en sortie. Par exemple, voici une fonction qui interverti l'ordre de deux nombres:

```
function [y1,y2]=mafonction(x1,x2)
%
% Définition de la fonction "mafonction":
%   arguments en entrée: x1 et x2
%   arguments en sortie: y1 et y2
%
  y1 = x2;
  y2 = x1;
```

Il s'agit ensuite d'appeler votre fonction à l'invite MATLAB:

```
>> [a,b]=mafonction(1,2)
a =
     2
b =
     1
```

Si on veut définir une fonction pour calculer x^2, on écrira

```
function y=carre(x)
%
% Définition de ma fonction x^2
%
  y = x*x;
```

En entrant help carre, vous verrez les lignes de commentaires qui suivent immédiatement le mot clef *function*:

```
>> help carre


  Definition de ma fonction x^2
```

Utilisez cette fonctionnalité pour vos fonctions.

Les communications entre les fonctions et les programmes se fait donc à l'aide des arguments en entrée et en sortie. La portée des variables définies à l'intérieur d'une fonction est donc limitée à cette fonction.

Cependant, dans certaines situations il peut être pratique d'avoir des variables globales, qui sont définies à l'aide de la commande *global*:

```
function y=Ccarre(x)
%
% Définition de la fonction (C^3) x^2
%
  global C          % variable globale, initialisée à l'extérieur de Ccarre
  e = 3             % variable locale a Ccarre, donc invisible ailleurs
  y = (C^e)*(x*x);
  C = 1             % C est modifiée partout ou elle est définie comme globale
```

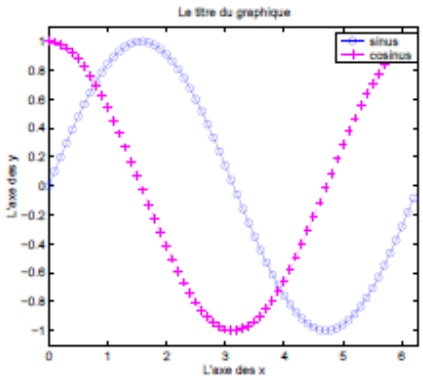
où la variable C serait aussi définie comme globale dans le *fichier-Moù* on veut y avoir accès. Par convention, les variables globales sont écrites en majuscules.

IX. Les graphiques

La fonction de base pour tracer un graphique avec MATLAB est la commande *plot*, qui prend comme arguments une série de points donnés sous la forme de 2 vecteurs, qu'elle relie de segments de droites. C'est la responsabilité de l'utilisateur de générer assez de points pour qu'une courbe régulière paraisse régulière à l'écran. De plus, il est possible de donner des arguments additionnels, ou d'utiliser d'autres fonctions, pour contrôler l'apparence du graphique:

```
>> x=0:0.1:2*pi;                          % L'ordonnée
>> plot(x,sin(x),'b-o',x,cos(x),'m--+');  % Le graphe de sinus et de cosinus
>> axis([0 2*pi -1.1 1.1]);               % Définition des axes
>> title('Le titre du graphique');
>> xlabel('L''axe des x');
>> ylabel('L''axe des y');
>> legend('sinus','cosinus');
```

Ces commandes donnent comme résultat:



On aurait aussi pu tracer ce graphique avec les commandes:

```
>> clf reset                  % on réinitialise l'environnement graphique
>> hold on
>> plot(x,sin(x),'b-o');      % un premier graphique
>> plot(x,cos(x),'m--+');     % on superpose un deuxième graphique
>> hold off
```

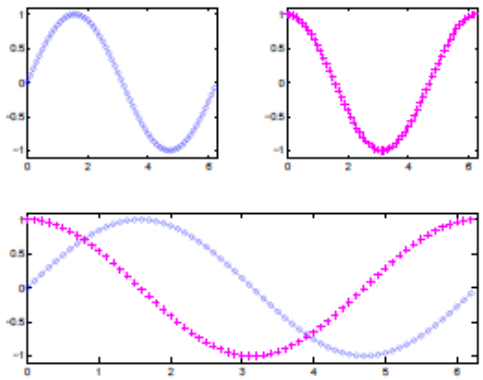
La commande *hold* évite que le premier graphique soit écrasé par le deuxième.

L'opérateur «*:*» n'est pas idéal pour créer le vecteur x. Comme vous pouvez le voir sur la figure précédente, x(end) n'est pas égal à 2_. Dans ce cas, on préfère utiliser la commande *linspace* pour générer le nombre de points voulu dans un intervalle donné, alors que l'opérateur « *:* » nous donne plutôt le contrôle sur la distance entre les points.

```
>> x1=0:1:2*pi                % combien a-t-on de points entre 0 et 2*pi?
x1 =
     0     1     2     3     4     5     6
>> x2=linspace(0,2*pi,7)      % ici on sait qu'on en a 7, allant de 0 à 2*pi
x2 =
         0    1.0472    2.0944    3.1416    4.1888    5.2360    6.2832
```

On peut aussi comparer des résultats sous forme graphique à l'aide de la commande *subplot*:

```
>> subplot(2,2,1); plot(x,sin(x),'b-o');axis([0 2*pi -1.1 1.1]);
>> subplot(2,2,2); plot(x,cos(x),'m-+');axis([0 2*pi -1.1 1.1]);
>> subplot(2,2,3:4);plot(x,sin(x),'b-o',x,cos(x),'m--+');
>> axis([0 2*pi -1.1 1.1]);
```



La fonction *fplot* facilite le tracé de graphes de fonctions, en automatisant le choix des points où les fonctions sont évaluées:

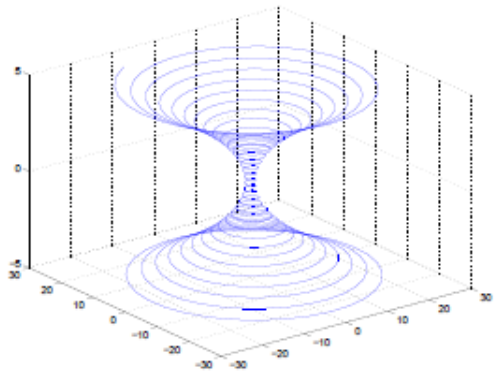
```
>> fplot('[sin(x),cos(x)]',[0 2*pi],'b-+')
```

On perd cependant de la latitude dans le choix de certaines composantes du graphique.

Finalement, les graphiques tridimensionnels, de type paramétrique, sont tracés à l'aide d'une généralisation de la commande *plot*:

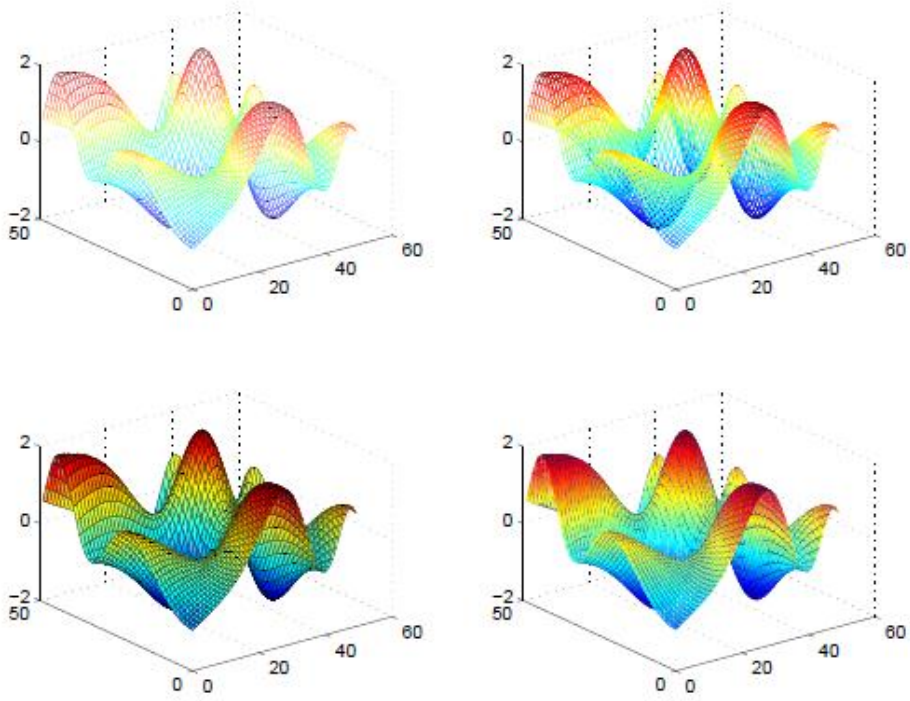
```
>> t=linspace(-5,5,1000);
>> x=(1+t.^2).*sin(20*t);     % utilisation de l'opérateur .^
>> y=(1+t.^2).*cos(20*t);
>> z=t;
>> plot3(x,y,z);
>> grid on;
```

qui donne comme résultat:



Pour ce qui est des graphes tridimensionnels, une étape intermédiaire est nécessaire avant d'utiliser les diverses commandes à notre disposition:

```
>> x=linspace(0,pi,50);
>> y=linspace(0,pi,50);
>> [X,Y]=meshgrid(x,y);          % on génère une grille sur [0,pi]x[0,pi]
>> Z=sin(Y.^2+X)-cos(Y-X.^2);
>> subplot(2,2,1); mesh(Z);
>> subplot(2,2,2); mesh(Z); hidden off;
>> subplot(2,2,3); surf(Z);
>> subplot(2,2,4); surf(Z); shading interp
```



X. Les sorties formatées

Nous aurons souvent à présenter des résultats numériques sous la forme de tableaux de nombres. Les programmeurs C ne seront pas dépaysés par les commandes MATLAB qui nous permettent de formater l'écriture dans des fichiers:

```
>> uns=(1:5)' ones(5,2)                          % création d'un tableau
uns =
     1     1     1
     2     1     1
     3     1     1
     4     1     1
     5     1     1
>> sortie=fopen('sortie.txt','w');               % ouverture du fichier
>> fprintf(sortie,'\t%s\n','Une serie de uns:'); % écriture de caractères,
>> fprintf(sortie,'\t%s\n','-----------------'); % d'entiers et de réels
>> fprintf(sortie,'\t%2d %4.2f %5.2e\n',uns');   % attention à la transposée
>> fclose(sortie);                               % fermeture du fichier
```

où la sortie dans le fichier *sortie*.txt sera:

```
>> type sortie.txt

        Une serie de uns:
        -----------------
         1 1.00 1.00e+00
         2 1.00 1.00e+00
         3 1.00 1.00e+00
         4 1.00 1.00e+00
         5 1.00 1.00e+00
```

Les caractères de contrôle pour le format de l'impression sont identiques à ceux que l'on retrouve dans le langage C. Dans l'exemple ci-dessus, «*\t*» représente une tabulation, «*\n*» un saut de ligne, «*%s*» une chaîne de caractères, «*%nd*» imprime un entier sur *n* caractères, «*%m.nf*» un réel sur *m* caractères avec *n* caractères après le point et «%*m.n*e» fait la même chose en notation scientifique.

Comme la plupart des commandes MATLAB, *fprintf* peut être utilisée sous forme vectorielle. Vous êtes donc encouragés à utiliser cette fonctionnalité dans vos programmes pour qu'ils s'exécutent plus rapidement.

Références bibliographiques :

- S.J. Chapman "MATLAB programming with applications for engineers" 2012.
- S. Dufour "Guide MATLAB " Court aide-mémoire pour les étudiants de MTH2210 École Polytechnique de Montréal 2004.
- H. Le-Huy "Introduction à MATLAB et Simulink' http://w3.gel.ulaval.ca/~lehuy/intromatlab/index.html
-