

## Travaux Pratiques numéro 2 :

### 1 Rappel sur les opérations

Effectuer les commandes suivantes et commenter les résultats obtenus (justifier pourquoi certaines commandes ne marchent pas correctement).

```
>> x = [1 2 3 4],  
>> y = [-1, 0, -6, 2],  
>> z = y',  
>> u = 3*x-y+2  
>> x - z,  
>> size(x),  
>> size(z),  
>> y./x,  
>> x./y,  
>> x*x',  
>> x'*x  
>> x/y,  
>> (x*y')/(y*y')  
>> A=[4 -1 2 0; 1 5 1 -2; -2 2 6 1; 1 0 -3 5],  
>> A*x,  
>> A*x',  
>> B = inv(A)  
>> B*x',  
>> A \ x',  
>> A(1,1),  
>> A(1,1:2),  
>> A(1:2,:),  
>> A^2,  
>> A.^2  
>> A'  
>> x = -1:0.1:1,  
>> plot(x,sin(x.^3),x,cos(x.^2))  
>> mma4 = ' méthodes informatiques ',  
>> length(mma4),  
>> mma4(1:9)  
>> z = 2 + 4 i,  
>> sqrt(z)
```

### 2 Pivot de Gauss et décomposition $LU$

- Comprendre chaque étape de l'algorithme qui suit.

- Programmer la fonction suivante

```
function x = GaussPivot(A,b)
% GaussPivot (A,b):
% Elimination de Gauss avec pivot partiel
% input :
% A = matrice
% b = vecteur
% output
% x = vecteur solution
[ m,n ] = size(A);
if m~=n, error ('Matrice A doit être carrée'); end
nb = n+1;
Aug = [A b];
% Elimination
for k = 1:n-1
% Pivot partiel
[ big,i ] = max(abs(Aug(k:n,k)));
ipr = i + k - 1;
if ipr ~= k
% Pivoter les lignes
Aug([ k,ipr],:) = Aug([ ipr,k ],:);
end
for i = k+1:n
factor = Aug(i,k)/Aug(k,k);
Aug(i,k:nb) = Aug (i,k:nb) - factor*Aug(k,k:nb);
end
end
% substitution pour revenir au problème initial
x = zeros(n,1);
x(n) = Aug(n,nb)/Aug(n,n);
for i = n-1:-1:1
x(i) = (Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
```

- Nous voulons résoudre un système linéaire à partir de ce programme.

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

- On peut utiliser aussi la commande `[L,U] = lu(A)`; . Expliquer le résultat obtenu.
- Prendre la matrice symétrique et trouver la décomposition de Choleski à l'aide de la commande

>> Help

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

### 3 Graphisme 2D

**Tracer le graphe d'une fonction; la commande `fplot`** La commande `fplot` permet de tracer le graphe d'une fonction sur un intervalle donné. La syntaxe est:

```
fplot('nomf', [xmin , xmax])
```

où *nomf* est soit le nom d'une fonction MATLAB incorporée, soit une expression définissant une fonction de la variable  $x$ , soit le nom d'une fonction utilisateur.  $[xmin, xmax]$  est l'intervalle pour lequel est tracé le graphe de la fonction.

Illustrons par des exemples les trois façons d'utiliser la commande `fplot`. On obtient le graphe de la fonction incorporée sinus entre  $-2\pi$  et  $2\pi$  par l'instruction:

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

Pour tracer le graphe de la fonction  $h(x) = x\sin(x)$  entre  $-2\pi$  et  $2\pi$ , on peut définir la fonction utilisateur  $h$  dans le fichier `h.m` de la manière suivante (attention de bien lire  $x.*\sin(x)$  et non pas  $x*\sin(x)$ ):

```
function y=h(x)
y=x.*sin(x);
```

On obtient alors le graphe de la fonction  $h$  par l'instruction:

```
>> fplot('h', [-2*pi 2*pi]).
```

L'autre façon de procéder est d'exécuter l'instruction (là on a le choix entre écrire  $x.*\sin(x)$  où  $x*\sin(x)$ ):

```
>> fplot('x*sin(x)', [-2*pi 2*pi]).
```

Il est également possible de gérer les bornes des valeurs en ordonnées. Pour limiter le graphe aux ordonnées comprises entre les valeurs  $ymin$  et  $ymax$  on passera comme second argument de la commande `fplot` le tableau  $[xmin, xmax, ymin, ymax]$ . 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([xmin, xmax, ymin, ymax])`.

```
>> fplot('sin(x)/x , cos(x)/x', [-5, 5, -1, 1])
>> axis([-5,5,-4,4])
```

On comprend très vite l'intérêt de gérer les bornes des valeurs en ordonnées si l'on exécute la commande `fplot('cos(x)/x', [-5, 5])` pour tracer le graphe de la fonction  $\cos(x)/x$  entre  $-5$  et  $5$ .

**La commande plot** La commande `plot` permet de tracer un ensemble de points de coordonnées  $(x_i, y_i)$ ,  $i = 1, \dots, N$ . La syntaxe est `plot(x,y)` où  $x$  est le vecteur contenant les valeurs  $x_i$  en abscisse et  $y$  est le vecteur contenant les valeurs  $y_i$  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  $(x_i, y_i)$  sont reliés entre eux par des segments de droites. Voici par exemple une autre façon de tracer le graphe de la fonction  $h(x) = x\sin(x)$  entre  $-2\pi$  et  $2\pi$ ,

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

Essayez aussi

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

**Légènder une figure** L'exemple suivant illustre l'utilisation des différentes commandes permettant de légènder une figure.

```
>> P = 5;
```

```

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

```

**La commande subplot** L'exemple suivant illustre l'utilisation de la commande `subplot`.

```

>> figure
>> subplot(2,3,1), fplot('cos',[0 4*pi]), title('cosinus'), grid
>> subplot(2,3,2), fplot('sin',[0 4*pi]), title('sinus'), grid
>> subplot(2,3,3), fplot('tan',[-pi/3 pi/3]), title('tangente'), grid
>> subplot(2,3,4), fplot('acos',[-1 1]), title('arc-cosinus'), grid
>> subplot(2,3,5), fplot('asin',[-1 1]), title('arc-sinus'), grid
>> subplot(2,3,6), fplot('atan',[-sqrt(3) sqrt(3)]), title('arc-tangente'), grid
>>

```

**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
  - `eps` : PostScript Encapsulé couleur
  - `jpeg` : Format d'image JPEG
  - `tiff` : Format d'image TIFF

## 4 Graphisme 3D

**Tracer les lignes de niveau d'une fonction de 2 variables** La commande `contour` permet de tracer les lignes de niveau d'une fonction de 2 variables réelles. Cette fonction peut être définie par une expression MATLAB (par exemple  $x \cdot \exp(-x^2 - y^2)$ ), ou être définie comme une fonction utilisateur. Pour tracer les lignes de niveau de la fonction  $g(x,y)$  pour  $x_{min} < x < x_{max}$  et  $y_{min} < y < y_{max}$  on procède de la manière suivante:

- création d'un maillage, de maille de longueur  $h$ , du domaine  $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$  grâce à la commande `meshgrid`,
 

```
[X,Y] = meshgrid(xmin:h:xmax, ymin:h:ymax).
```

- évaluation de la fonction aux noeuds de ce maillage, soit par appel à la fonction utilisateur définissant la fonction,

`Z = g(X,Y)`

soit directement en définissant la fonction par une expression MATLAB.

- Affichage des lignes de niveau grâce à la commande `contour`,  
`contour(X,Y,Z)`.

Ainsi pour tracer les lignes de niveau de la fonction  $f(x, y) = xe^{-(x^2+y^2)}$  sur le domaine  $[-2, 2] \times [-2, 2]$  en prenant un maillage de maille de longueur  $h = 0.2$ , on exécute:

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = X.*exp(-X.^2-Y.^2);
>> contour(X,Y,Z)
>>
```

On peut également écrire une fonction utilisateur *g.m*,

```
function x3 = g(x1,x2)
x3 = x1.*exp(-x1.^2-x2.^2);
```

et exécuter

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = g(X,Y);
>> contour(X,Y,Z)
>>
```

Le nombre de lignes de niveau est déterminé de manière automatique à partir des valeurs extrêmes prises par la fonction sur le domaine considéré. Pour imposer le nombre  $n$  de lignes de niveau à afficher, il suffit d'appeler la fonction `contour` avec la valeur  $n$  comme quatrième paramètre,

```
>> contour(X,Y,Z,n).
```

Il existe deux manières d'afficher les valeurs des lignes de niveau sur la figure. Si l'on souhaite afficher les valeurs pour toutes les lignes de niveau, on utilise la commande `clabel` de la manière suivante:

```
>> [C,h] = contour(X,Y,Z,n)
>> clabel(C,h)
```

Si l'on souhaite afficher uniquement les valeurs de quelques lignes de niveau, on utilise la commande `clabel` de la manière suivante:

```
>> [C,h] = contour(X,Y,Z,n)
>> clabel(C,h, 'manual')
```

On peut alors grâce à la souris sélectionner les lignes de niveau pour lesquelles on souhaite afficher la valeur.

Ainsi pour tracer 30 lignes de niveau de la fonction  $z = (x-1)^2 + 10(x^2-y)^2$  sur le domaine  $[-1, 1] \times [-1, 1]$  et pour choisir à l'aide de la souris les lignes de niveau pour lesquelles l'isovaleur doit être affichée, on exécute:

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = (X-1).^2 + 10*(X.^2-Y).^2;
>> [C,h] = contour(X,Y,Z,30);
```

```
>> clabel(C,h,'manual')
>>
```

**Modifier la couleur des courbes.** Il est possible de modifier la palette des couleurs en utilisant la commande `colormap`. En tapant `help graph3d` dans la fenêtre de contrôle MATLAB, vous obtiendrez toutes les palettes de couleurs disponibles. À vous de les tester pour obtenir le meilleur effet. La commande `colormap(gray)` permet d'utiliser une palette en dégradé de gris, ce qui est très utile si l'on souhaite une impression de la figure sur une imprimante noir et blanc.

La commande `contourf` s'utilise de la même manière que la commande `contour`. Elle permet d'afficher, en plus des lignes de niveau, un dégradé continu de couleurs qui varie en fonction des valeurs prises par la fonction.

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = (X-1).^2 + 10*(X.^2-Y).^2;
>> contourf(X,Y,Z,30);
>> colormap(cool);
>>
```

**Représenter une surface d'équation  $z=g(x,y)$**  La commande `mesh` permet de tracer une surface d'équation  $z = g(x, y)$ . La fonction  $g$  peut être définie directement par une expression MATLAB ou être définie comme une fonction utilisateur. Pour tracer la surface d'équation  $z = g(x, y)$  pour  $x_{min} < x < x_{max}$  et  $y_{min} < y < y_{max}$  on procède de la manière suivante:

- création d'un maillage, de maille de longueur  $h$ , du domaine  $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$  grâce à la commande `meshgrid`,  

```
>> [X,Y] = meshgrid(xmin:h:xmax, ymin:h:ymax).
```
- évaluation de la fonction aux noeuds de ce maillage, soit par appel à la fonction utilisateur définissant la fonction,  

```
>> Z = g(X,Y)
```

soit directement en définissant la fonction par d'une expression MATLAB.
- affichage de la surface grâce à la commande `mesh`  

```
>> mesh(X,Y,Z)
```

Ainsi pour tracer la surface d'équation  $z = xe - (x^2 + y^2)$  sur le domaine  $[-2, 2] \times [-2, 2]$  avec un maillage de maillage de longueur  $h = 0.2$ , on exécute:

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = X.*exp(-X.^2-Y.^2);
>> mesh(X,Y,Z)
>>
```

Si la fonction est définie comme une fonction utilisateur dans le fichier `g.m`,

```
function x3 = g(x1,x2)
x3 = x1.*exp(-x1.^2-x2.^2);
```

on exécute:

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = g(X,Y);
>> contour(X,Y,Z)
>>
```