Jean-Claude Yakoubsohn

Guide d'utilisation du logiciel MATLAB

Laboratoire MIP, Département de Mathématiques
Université Paul Sabatier

31062 Toulouse cedex 04
yak@mip.ups-tlse.fr

1. L'environnement MATLAB

MATLAB est la contraction de *matrix laboratory*. C'est un logiciel de calcul scientifique conçu par une équipe d'ingénieurs et universitaires américains: il est au calcul numérique ce que Maple est au calcul formel. Il est spécialement développé autour du calcul matriciel, la résolution d'équations différentielles, l'intégration et la représentation graphique. De plus il est possible d'écrire ses propres applications et de les référencer de manière identique aux fonctions classiques de MATLAB. Pour toutes ces raisons MATLAB est un outil d'expérimentation, de simulation et de mise au point d'algorithmes trés intéressant.

1.1 Démarrer et quitter MATLAB

Sous UNIX

Il suffit faire

matlab

dans une fenêtre xterm. La fenêtre MATLAB s'ouvre.

Pour quitter MATLAB faire

quit

dans la fenêtre de MATLAB.

Sous Windows

Un double-click sur l'icone MATLAB ouvre la fenêtre de travail.

L'**Environnement MATLAB** comprend :

- une fenêtre de commande
- un éditeur
- un historique des commandes
- un aide en ligne
- un "profiler" calculant la performance des calculs.

On renvoie à l'aide MATLAB pour les différentes options de configuration de toutes ces fenêtres.

1.2 Travailler dans la fenêtre de commande

La fenêtre de commande n'est pas un éditeur pleine page. Le curseur n'a pas la possibilité d'aller vers le haut. Son fonctionnement est identique au système de commandes UNIX: une commande suivie de return implique une réponse. Ce n'est pas un problème puisque les commandes effectuées sont stockés dans un "buffer": ce qui permet de les rappeler au clavier avec ↑ ou **Ctrl P**. Les commandes d'édition de lignes sont celles de l'éditeur emacs.

Une commande UNIX s'exécute depuis MATLAB : faire !commande-UNIX.

Remarque importante: les minuscules et les majuscules sont différenciées dans MATLAB.

1.3 Premier pas

1.3.1 Assignation et affichage

Le prompt de MATLAB est

$$>>$$

Toute commande MATLAB s'effectue aprés ce >>. Ainsi pour définir une matrice A , 2×3 il suffit de faire

```
>> A=[ 1/3 2 3;4 5 6]; % ceci est une matrice
```

Aprés avoir presser sur la touche **Entrée**, la réponse de MATLAB est

```
A =

     1     2     3
     4     5     6
```

Le symbole pour cent

%

est optionnel : il est utilisé pour commenter les commandes

Par contre en ajoutant un point virgule

;

après la commande MATLAB, il n'y a pas d'affichage du résultat. Mais la valeur de la matrice A est celle qui lui a été assignée.

Il est donc important de ne pas oublier les points virgules à la fin des commandes que l'on ne veut pas afficher.

Tous les calculs sont effectués en double précision. Il existe plusieurs type de format d'affichage :

```
1- format short
2- format long
3- format rat
```

Ainsi

```
format short
a=1.2345678987654356
```

donne

```
a =

  1.2346
```

Tandis que

```
format long
a=1.2345678987654356
```

donne

```
a =

  1.23456789876544
```

Et

```
format rat
a=1.2345678987654356
```

donne

```
a =

  100/81
```

Dans les trois précédents la valeur affichées n'est pas celle assignée.

L'affichage des résultats utilise le

```
>> format loose
```

Un autre format plus pratique supprime la ligne d'espace dans l'affichage par défaut. Les commandes

```
>> format compact
>>A=[1 2 3; 4 5 6]
```

affichent

```
A =
     1     2     3
     4     5     6
```

Il est possible d'écrie plusieurs fonctions sur une même ligne et d'aller à ligne par **Maj Entré**. On utilise alors une virgule ou un point virgule entre les commandes suivant qu'on veuille afficher ou non les résultats. Les commandes

```
>>format compact
  % x et y s'affichent au format short, z ne s'affiche pas
  format short, x=2:3, y=log(x), z=exp(x);
  % y  et z s'affichent au format long
  format long   y=log(x),z=exp(x)
```

donnent

```
x =
     2     3
y =
    0.6931    1.0986
y =
   0.69314718055995   1.09861228866811
z =
   7.38905609893065  20.08553692318767
```

Si une commande ne tient pas sur une ligne, il suffit de le signaler par

...

La commande

```
A=[123456789, ...
987654321]
```

donne

```
A =
   123456789   987654321
```

1.3.2 Aide en ligne

MATLAB possède plusieurs milliers de commandes. Il y a un aide et un index trés bien documenté pour s'y retrouver. Ainsi la commande

```
>> help lu
```

donne tous renseigements sur la factorization LU d'une matrice. Le *help* de MATLAB est l'équivalent *man* d'UNIX.

Pour accéder à l'index on utilise

```
>>lookfor
```

Par exemple:

```
>> lookfor inverse
```

donne les thèmes, fonctions et commandes qui touchent de près ou de loin à ce mot inverse qui n'est pas une fonction MATLAB. C'est l'équivalent de apropos dans UNIX.

Pour exécuter un programme MATLAB (M-file) il suffit de l'appeler par son nom avec ses arguments. La commande

```
>> A=[1 2 3 ; 4 5 6 ; 6 7 8];
   eigs(A)
```

calcule les valeurs propres de la matrice A. L'affichage donne avec les format compact et format long

```
 ans =
  16.11684396980704
  -1.11684396980704
  -0.00000000000000
```

Il est parfois utile de voir comment est programmer une fonction MATLAB. La commande

```
 >> edit eigs
```

édite le code MATLAB de la fonctionn eigs.

1.3.3 Une première fonction et un premier graphique

Pour créer ses propres fonctions (M-file) on ouvre l'éditeur MATLAB. On édite la fonction en question et on sauve le fichier texte dans le répertoire courant avec l'extension .m sous la forme

nom_de_fichier.m

Pour exécuter on saisit le nom de la fonction avec ses arguments dans la fenêtre de commandes. Voici un exemple de fonction qui représente une astroïde et qui donne l'équation de la courbe en sortie. Le texte de la fonction est

```
function equation=astroide(a,b)

c=a^2-b^2;
equation=['t-->[' num2str(c/a) 'cos(t)^3,', '-' num2str(c/b) 'sin(t)^3]'];
t=0:10^(-3):2*pi;
plot(c/a*cos(t).^3,c/b*sin(t).^3,'r')
```

La commande

```
astroide(3,2)
```

affiche

```
ans=
t-->[1.6667cos(t)^3,-2.5sin(t)^3]
```

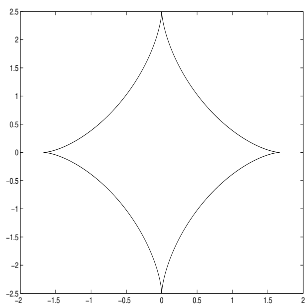
Commentaires

1- Le nom de la fonction est astroïde. Les arguments d'entrées sont a, b. L'argumennt de sortie est equation.

2- La quatrième ligne de la fonction définit l'équation paramétrée à l'aide de la fonction num2str qui convertit un nombre en une chaîne de caractères.

3- La sixième ligne utilise **plot** pour dessiner la courbe. Un paragraphe est consacré au graphique.

Dans le même temps une nouvelle fenêtre s'ouvre et produit le graphe de l'astroïde.



Pour charger un fichier nom.m faire

load 'nom'

1.4 Facilités

Les commandes suivantes sont utiles:

clear : efface variables et fonctions de la mémoire.

disp : écriture d'une matrice.

clc : commande *clear* pour la fenêtre.

who : donne la liste des variables courantes.

clf : efface la figure dans la fenêtre graphique.

1.5 Nombres, Matrices et Opérations

Les **nombres** sont représentés en base 2 avec 54 chiffres significatifs. Ce qui correspond à 16 chiffres significatifs en base 10. Le domaine des nombres flottants représentables est $[10^{-308}, 10^{308}]$. Les opérateurs arithmétiques entre nombres ou matrices sont:

$$+ \quad - \quad * \quad / \quad \backslash \quad \hat{}$$

On peut remarquer qu'il y a une division à droite et à gauche :1/4 et 4\1 donnent le même résultat 0.25.

L'**assignation** se fait par le signe

$$=$$

Si le point virgule ; apparaît aprés l'assignation le résultat ne s'affiche pas à l'écran. Il n'y a pas d'affichage avec commande

```
x=0.5;
```

Il y a affichage si aucun signe n'est utilisé aprés l'assignation:

```
x=0.5
```

L'infini est produit par 1/0 qui donne inf. Quand un nombre n'est pas dans le domaine des flottants définis par MATLAB il prend la valeur not a number :

nan

Il y a des variables et des constantes spécifiques dont les plus courantes sont:

ans : réponse la plus proche.

computer : type de machine.

cputime : temps cpu.

eps : précision de l'arithmétique flottante (IEEE eps=2.2×10^{-16}).

flops : nombre d'opérations flottantes.

i,j : imaginaire pur (peut être utilisé par ailleurs mais non conseillé).

nargin : nombre d'arguments en entrée d'une fonction.

nargout : nombre d'arguments en sortie d'une sortie d'une fonction.

pi : ce nombre utile aux sages...

realmax : plus grand nombre flottant. (1.797693134862316e+308)

realmin : plus petit nombre flottant.(2.225073858507201e-308)

Les **matrices** constituent les objets autour desquelles est structuré MATLAB. Le jeu des notations pour les créer, les représenter et les manipuler est à la fois simple et synthétique.

Une matrice est un tableau $n \times m$. Les opérateurs arithmétiques sont

```
+   Addition
-   Suoustraction
*   Multiplication
/   Division
.*  Multiplication composants par composants
./  Division composants par composants
^   Puissance
.^  Puissance composants par composants
.'  Transposition
'   Transosition-Conjugaison
:   Opérateur colonne
```

Ainsi les opérations entre matrices sont celles de l'algèbre linéaire. Sous la condition qu'elles soient licites on écrit:

- A*B : multilication matricielle de A par B.
- A/B synonyme de A*inv(B) et $A\backslash\ B$ synonyme inv(A)*B.
- A\p : puissance $p^{ème}$ d'une matrice.
- A.' : transposée de A.
- A' : transposée-conjuguée de A.

On prendra soin de ne pas confondre

```
A=[1-2*i 2+i;3+2*i 5]'
```

qui donne

```
A =
      1       +    2i             3       -    2i
      2       -    1i             5
```

et

```
A=[1-2*i 2+i;3+2*i 5].'
```

qui donne

```
A =
      1       -    2i             3       +    2i
      2       +    1i             5
```

Ceci est une source d'erreur. Ainsi si on veut calculer la transposée d'une matrice, il est conseillé d'écrire A.' plutôt que A' et ceci principalement dans une fonction.

En plus de ces opérations, il y a les opérations :

$$.* \qquad ./ \qquad .\backslash \qquad .\hat{}$$

Elles sont définies par

$$\mathsf{A.}\epsilon\ \mathsf{B\ est\ la\ matrice} \quad (A_{kl}\,\epsilon\,B_{kl}) \quad \mathsf{où}\ \epsilon \in \{*, \quad \backslash, \quad /, \quad \hat{}\}.$$

Par exemple

```
A=[2 3;4,5];   B=[6,1;3,-2];
A.*B
```

donne

```
ans =
      12            3
      12          -10
```

Tandis que

```
A=[2 3;4,5];   B=[6,1;3,-2];
A.^B
```

donne

```
ans =
     21           -4
     39           -6
```

1.6 Manipulation des matrices

La commande

```
A=[1:4;5:8;9:12]
```

donne

```
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Pour accéder au coefficient d'indice (k,l) il suffit de faire A(k,l).Les indices des coefficients sont toujours plus grands ou égaux à 1. L'écriture A(0,l) n'a pas sens.

Les manipulations de matrices sont basées sur les usages suivants:

A(k,l) est l'élément k,l de A. MATLAB stocke les éléments d'une matrice colonne par colonne sous la forme d'un vecteur colonne. Ainsi

A(n(l-1)+k) est identique à A(k,l)

A(i1:i2 , j1:j2) est le bloc extrait de la matrice A comprenant les lignes i_1 à i_2 et les colonnes j_1 à j_2.

A(:,:) est la matrice A

sf A(k,:) est la ligne k de A

A(:,l) est la colonne l de A.

A(:) est la matrice A transformée en une simple colonne.

A(k,l:end) : les éléments d'indice plus grand que l de la ligne k.

De façon générale l'utilisation du signe

:

règle la manipulation des indices:

k:l est identique à $[k, k+1, \dots, l]$ si $l \geq k$ et [ ] (matrice vide) sinon.

k:l:m est idendique à $[k, k+l, k+2l, \dots, m]$ si $l > 0$ et $m > k$ ou $l < 0$ et $m < k$ et [ ] sinon. Ainsi

```
1:10,  1:3:10, 10:-2:1, 1:-1
```

donne successivement

```
ans =
     1          2          3          4          5          6
ans =
```

```
      1            4            7           10
ans =
     10            8            6            4            2            0
ans =
   Empty matrix: 1-by-0
```

Size

La taille d'une matrice est donné par

$$[m,n]=size(A)$$

Ainsi

```
A=[1,3,2];   [m,n]=size(A)
```

donne

```
m=
   1
n=
   3
```

Reshape

La commande reshape permet de retailler une matrice. Par exemple de transformer une matrice en un vecteur. De façon générale

$$reshape(A,m,n)$$

retourne une matrice de taille $m \times n$ ayant les mêmes éléments que la matrice A. Si la matrice A ne possède pas mn éléments un message d'erreur en résultera.

```
A=[1:4;5:8;9:12];
reshape(A,2,6)

ans =
      1          9          6          3         11          8
      5          2         10          7          4         12
```

1.7 Matrices spéciales.

[] : matrice vide.

zeros : matrice nulle.

ones : matrice dont tous les éléments sont égaux à 1.

eye : matrice identité.

rand : matrice dont les éléments sont uniformément distribués.

randn : matrice dont les éléments sont normalement distribués.

On a aussi à disposition un jeu de matrices classiques:

compan : matrice compagnon.

diag : matrice diagonale.

gallery : jeu de matrices tests.

hadamard : matrice de Hadamard.

hilbert : matrice de Hilbert.

invhilb : matrice Hilbert inverse.

kron : produit tensoriel de Kronecker.

magic : carré magique.

pascal : triangle de Pascal.

toeplitz : matrice de Toeplitz.

vander : matrice de Vandermonde.

wilkinson : matrice de Wilkinson.
classique.

1.8 Analyse matricielle:

cond : conditionnement d'une matrice.

det : déterminant.

norm : norme d'un vecteur ou d'une matrice.

null : noyau.

orth : orthogonalisation.

rcond : conditionnement inverse.

rank : rang.

trace : trace.

1.9 Systèmes linéaires:

chol : factorisation de Choleski.

inv : matrice inverse.

lscov : moindres carrés avec covariance donnée.

lu : factorisation LU.

nnls : moindres carrés positifs.

pinv : pseudoinverse.

qr : décomposition QR.

\, / : solution de systèmes linéaires.

1.10 Valeurs propres et valeurs singulières:

balance : mise à l'échelle pour améliorer la précision du calcul des valeurs propres.

cdf2rdf : passage de la forme diagonale complexe à la forme réelle diagonale par blocks.

eig : calcul des valeurs propres.

hess : forme de Hessenberg.

poly : polynôme charactéristique.

qz : valeurs propres généralisées.

rsf2csf : passage de la forme réelle diagonale par blocks à la forme diagonale complexe.

schur : décomposition de Schur.

svd : décomposition en valeur singulière.

Fonctions de Matrices:

expm : exponentielle de matrice.

expm1 : exponentielle de matrice vi approximation de Padé.

expm2 : exponentielle de matrice via le développement de Taylor.

expm3 : exponentielle de matrice via les valeurs propres et les valeurs singulières.

funm : évaluation générale de fonction de matrice.

logm : logarithme de matrice.

sqrtm : racine carré de matrice.

1.11 Relations et opérateurs logiques

Les relations suivantes sont licites entre nombres vecteurs ou matrices de dimensions identiques:

< <= > >= == ~=

La dernière relation signifie "non égal à". Par exemple:
>> 2+2~= 4 donne 0.
>> 2+2==4 donne 1.
Les opérateurs logiques sont:

& : et.

| : ou.

~ : négation.

xor : ou exclusif.

La valeur d'une opération logique est 1 ou 0 suivant qu'elle est vraie ou fausse.

A & B est la matrice dont les éléments sont 1 ou 0 suivant que:

$$c_{ij} = (a_{ij} \sim= 0)\ \&\ (b_{ij} \sim= 0)$$

A | B est la matrice C dont les éléments sont 1 ou 0 suivant que:

$$c_{ij} = (a_{ij} \sim= 0) \mid (b_{ij} \sim= 0)$$

~A : matrice C formé de 1 ou 0 suivant que:

$$c_{ij} = a_{ij} == 0$$

Any, all

any : any(x) est 1 si un des éléments de x est non nul et 0 sinon. Ainsi

```
A=[1 2 3]
A<3
any(ans)
```

donne

```
 A =
      1             2             3
ans =
      1             1             0
ans =
      1
```

all : all(x) est 1 si tous les éléments de x sont nuls et 0 sinon. Ainsi

```
A=[1 2 3]
A<3
all(ans)
```

donne

```
 A =
      1            2            3
ans =
      1            1            0
ans =
      0
```

Ces fonction sont tr'ès utiles pour réaliser des tests du genre:

```
  if all(A <.6) }
         déclarations
  end
```

Les principaux opérateurs et relations logiques sont:

find : tableau d'indices de valeurs logiques.

exist : recherche si une variable existe.

isnan : détecte NAN

isinf : détecte "l'infini".

finite : vraie pour toute valeur finie.

isempty : vraie pour la matrice vide.

Exemple: les commandes

```
A=[1:4;5:8;9:12]; k=find( $A>=8$),  A(k)=2zeros(size(k))
```

remplacent les éléments de la matrice A plus grands que 8 par 0.

```
k =
     3
     6
     9
    11
    12
A =
     1     2     3     4
     5     6     7     0
     0     0     0     0
```

Fonctions Mathématiques.

Les fonctions trigonométriques:

sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh, asinh, acosh, atanh.

Les fonctions élémentaires sont:

abs : valeur absolue ou module suivant le contexte.

angle : argument en radians compris entre $-\pi$ et π.

sqrt : racine carré.

real : partie réelle.

imag : partie imaginaire.

conj : complexe conjuguée.

round : arrondi à l'entier le plus proche.

fix : arrondi par rapport à zéro.

floor : arrondi par rapport à $-\infty$.

ceil : arrondi par rapport à $+\infty$.

sign : fonction signe.

rem : reste par division euclidienne.

gcd : pgcd.

lcm : ppcm.

exp : exponentielle.

log : logarithme en base e.

log10 : logarithme en base 10.

Toutes les fonctions précédentes s'appliquent aux matrices:

par exemple sin(A) est la matrice $(sin(A(i,j)))$.

On trouve aussi des fonctions plus compliquées:

bessel, beta, gamma, rat, erf, erfinv, ellipk, ellipj.

1.12 Graphique

Le graphique de Matlab est construit à partir d'objet élémentaires: lignes, surfaces, etc... Comme pour le calcul matriciel, des fonctions permettent de manipuler ces objets de base. Dans un premier temps Nous n'abordons que le graphique élémentaire.

1.12.1 Graphique 2-D

Si y est un vecteur plot(y) produit un graphique dont les points (i, y_i), $(i+1, y_{i+1})$ sont reliés par des segments de droites.

Par contre plot(x,y) produit un graphique dont les points (x_i, y_i), (x_{i+1}, y_{i+1}) sont reliés par des segments de droites.

plot : création d'un graphique de vecteurs ou colonnes de matrices.

loglog : graphique avec échelle logarithmique sur chacun des axes.

semilogx : graphique avec échelle logarithmique sur l'axe des x seulement.

semilogy : graphique avec échelle logarithmique sur l'axe des y seulement.

Les options suivantes sont possible:

linewidth : épaisseur du trait

title : ajoute un titre.

legend : ajoute une légende

xlabel : ajoute une étiquette à l'axe des x.

ylabel : ajoute une étiquette à l'axe des y.

text : ajoute un texte à un endroit spécifié.

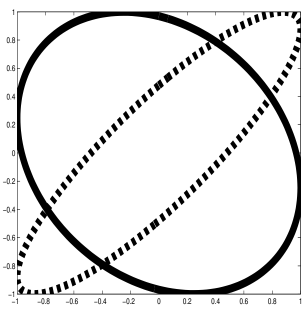
gtext : ajoute un texte à l'aide de la souris.

Il est posible de représenter plusieurs graphiques avec des couleurs et des styles distincts.

Exemple:

```
t=0:pi/100:2*pi;  x=sin(t);
y1=cos(t+.25);    y2= sin(t+.5);
plot(x,y1,'r-',x,y2,'g--','linewidth',10)
```

Deux ellipses une en rouge (red) en trait plein et l'autre en vert (green) d'épaisseur 10 seront représentées.



Les caractères admis pour le tracé sont:

$$. \quad \times \quad + \quad * \quad - \quad : \quad -- \quad -. \quad \circ$$

Les couleurs sont:

y (yellow), m (magenta), c (cyan) , r (red), g (red), b (blue), w (white), k (black).

Des courbes distinctes peuvent être représentées sur la même figure de la façon suivante. Avec les courbes ci-dessus il suffit de faire:

```
plot(x)
hold on
plot(y1,'-')
hold off
```

Une fonction se représente aussi en faisant:

```
x=(0:1/2000:1)';  plot(x,cos(tan(pi*x)))
```

1.12.2 Graphique 3-D

Les fonctions de représentation de courbes et surfaces 3-D sont:

plot3 : représente lignes et points.

contour, contour3 : création de contour.

pcolor : dessine un tableau rectangulaire de cellules dont les couleurs sont déterminées par les éléments d'une matrice.

image : représentation une matrice par une image.

meshgrid : création d'un maillage.

mesh, meshc, meshz : représentation d'une surface décrite matriciellement par un maillage.

surf, surfc, surfl : représentation d'une surface décrite matriciellement par des facettes.

fill3 : construit un polygone et colorie ses faces.

xlabel,ylabel,zlabel,clabel : création de label pour les axes et pour les contours.

view : choix du point de vue.

viewmtx : matrice 4×4 de transformation du point de vue.

On se reportera à l'aide en ligne pour toutes ces fonctions.

Pour transformer une figure en fichier postscript faire

print -dps name.ps

Exemple de représentation d'un paraboloide elliptique. Soit A une matrice 2×2 définie positive et b un vecteur du plan. Le graphe de la fonction

$$J(x) = x'Ax + b'x + c$$

est un paraboloide elliptique. On se propose de représenter le graphe

$$G = \{(x, J(x)) \in D \times \mathbf{R}, \quad \}$$

où D est un rectangle du plan. Le principe est de définir un maillage du rectangle D et d'évaluer J en chaque point du maillage. La fonction meshgrid permet la définition du maillage et la fonction mesh représente une surface passant par les points $(x, J(x)$ calculés.

```
A=[3 1;1 2];b=[-1,1]';c=4;
[X,Y]=meshgrid([-2:.1:2],[-3:.1:1]);
J=1/2*X.^2*A(1,1) + (X.*Y)*A(1,2) + 1/2*Y.^2*A(2,2) - X*b(1) - Y*b(2)+ c;
mesh(X,Y,J)
```

1.13 Boucles et Tests

1.13.1 Boucle for

```
for k=k1:pas:k2$ ,
    A(1,k)=2*k;
end
```

avec les conventions décrites ci-dessus concernant k1:pas:k2

Généralement

```
for k= expression
    commandes
end
```

L'index d'une boucle peut être un tableau.

```
for k=A
    commandes
end
```

Á l'étape k, alors k=A(:,k). Ainsi l'exécution de

```
for n= [1 2 3;4 5 6]
    n
end
```

donne

```
 n =
     1
     4
n =
     2
     5
n =
     3
     6
```

1.13.2 Boucle while

i=0; while i¡=100 , i=i+1; end

Généralement

```
while expression
    commandes
end
```

1.13.3 if, else, elseif

```
if  condition
   commandes
elseif  condition
         commandes
else
   commandes
end
```

1.13.4 continue

Cette commande permet de passer à l'étape suivante dans boucle for ou while en sautant des commandes dans le corps de la boucle. Voici un exemple qui compte le nombre de lignes de code dans la fonction eigs.m. Les commandes

```
f = fopen('eigs.m', 'r');
nl = 0;
while ~feof(f)
     ligne = fgetl(f);
     if isempty(ligne) | strncmp(ligne, '%', 1)
           continue
      end
      nl = nl + 1;
end
```

```
disp(sprintf('%d lignes', nl));
fclose(f);
```

affichent

```
897 lignes
```

Remarques :

- La commande feof retourne 1 si on est à la fin du fichier et 0 sinon.
- fgetl retourne la ligne suivante dans le fichier.
- strncmp(s1, s2, n) retourne 1 si les n premiers charactères de s1 et s2 sont identiques et 0 sinon.

1.13.5 break

La commande break stoppe l'exécution d'une boucle. L'exécution du programme continue apré la boucle.

L'exemple suivant montre l'algorithme définissant la commande **help** de MATLAB.

```
f = fopen('sign.m', 'r');
s = '';
while ~feof(f)
     ligne = fgetl(f);
     if isempty(ligne)
        break
     end
     s = strvcat(s, line);
end
disp(s)
```

Ces commandes affichent

```
%SIGN   Signum function.
%   For each element of X, SIGN(X) returns 1 if the element
%   is greater than zero, 0 if it equals zero and -1 if it is
%   less than zero.  For the nonzero elements of complex X,
%   SIGN(X) = X ./ ABS(X).
```

1.13.6 return

Cette commande force l'exécution d'une fonction.

1.13.7 switch

```
switch expression scalaire ou charactère
      case valeur_1
          commandes
      .
      .
      case valeur_n
           commandes
      otherwise
           commandes
end
```

Dés qu'un des case est satisfait les autres ne s'exécutent pas.

1.13.8 error

La commande

error('texte signalant une erreur')

provoque une interruption de l'exécution d'un M-fichier.

M-file: Texte et Fonctions

Un **texte** Matlab est une suite de déclarations MATLAB. Il est conseillé de regarder la boite à outils *demos* pour apprendre à écrire clairement une suite de déclarations MATLAB.

Une **fonction** MATLAB est un texte dont la première ligne est

function output = nom_de_fonction(argument1,argment2,...)

Le fichier contenant la fonction MATLAB poss'ede l'extension .m: il est fortement conseillé de le nommer nom.m.

Exemple:

```
function [g,e]=barycentre(A)
% Calcul
       % 1-le barycentre g de n vecteurs de taille m avec [m,n]=size(A).
       % 2- la moyenne e de tous les éléments de la matrice A.
[m,n]=size(A);
g=(:);
for j=1:n , g(j)=sum(A(:,j))/m; end
e=sum(g)/n;
```

Si A est la matrice A=[0:1:3; 4:2:10; 3:3:12]. Les commandes [g,e]=barycentre(A); g,e

donnent

```
g=
  [2.33,4.33,6.33]
e=
  3.75
```

La commande help barycentre donne

```
  Calcul
        1-le barycentre g de n vecteurs de taille m avec [m,n]=size(A).
        2- la moyenne e de tous les éléments de la matrice A.
```

En conclusion on a augmenté la bibliothèque MATLAB d'une nouvelle fonction. Les fichiers de fonctions sont construits sur le schéma suivant:

1- La première ligne déclare le nom de la fonction, les arguments d'entrée et les arguments de sorties. C'est cette ligne qui distingue un texte d'une fonction. Elle est inclue dans le fichier d'index de MATLAB Contents.m. Elle sera activée si nécessaire par la commande lookfor.

2- Le symbole % est facultatif: il indique les lignes de commentaires qui seront activées par la commande help.

3- Les variables du fichier nom.m sont locales à moins de les déclarer comme globales dans nom.m avec la commande global.

4- L'emploi des variables nargin et nargout est utilisé dans la construction des M-files.

En effet le nombre d'arguments en entrée et en sortie d'une fonction peut varier. Typiquement une fonction s'écrit

```
function varargout=nom_de_fonction(varargin)
Tests sur nargin et/ou nargout
commandes
varargout=expression
```

Ainsi la fonction *eigs* s'écrit (pour le voir faire *edit eigs*):

```
function  varargout = eigs(varargin)
%EIGS  Find a few eigenvalues and eigenvectors of a matrix using ARPACK.
%   D = EIGS(A) returns a vector of A's 6 largest magnitude eigenvalues.
%   A must be square and should be large and sparse.
.
.
.
```

5- D'un point de vue pratique la définition d'une fonction passe par la création d'un fichier nom.m. Ici il y a une différence avec un fichier C ou Fortran où les procédures et fonctions peuvent coexister dans le même fichier.

Il est cependant un cas où plusieurs fonctions peuvent être dans le même fichier: c'est celui des sous-fonctions. La première fonction du fichier est la fonction principale. Les autres sont des sous-fonctions. Elles ne peuvent pas être appelées par une fonction extérieure à ce fichier. De plus la fonction help n' accède qu' à la fonction principale.

```
function [g,e]=barycentre(A)
% Calcul
       % 1-le barycentre g de n vecteurs de taille m avec [m,n]=size(A).
       % 2- la moyenne $e$ de tous les éléments de la matrice A.

[m,n]=size(A);
g=(:);
for j=1:n , g(j)=sum(A(:,j))/m; end
e=moyenne(g,n)

function emoyenne(g,n)
e= sum(g)/n;
```

6- Pour utiliser une fonction ne faisant partie de MATLAB de base il faut qu'elle soit dans le même répertoire d'activation de MATLAB.

1.14 Temps CPU et allocation mémoire

La commande cputime donne le temps d'exécution depuis le début de la session. Il faut noter que le temps d'exécution de certaines commandes est important. Notamment le temps d'exécution des boucles. Il faut toujours opter pour les versions vectorisées des taches à accomplir quand celà est possible. Par exemple on préferera:

```
t_sans_boucle=cputime;
t=0.:a:10.; y=sin(t);
tt=cputime-tt
```

à

```
t_avec_boucle=cputime;
y=[]; i=0;
for t=0.:a:10
      i=i+1; y(i+1)=sin(t);
end
```

On obtient pour $a = 10^(-5)$ des temps cpu : t_avec_boucle=82.7 et t_sans_boucle=0.01 Il y a un facteur 8000 entre les temps cpu des deux taches ci-dessus. Il est préférable d'allouer préalablement les vecteurs avant l'exécution d'une boucle. Par exemple:

```
y=zeros(100,1);
for i=1:100
    y(i)=sin(2*pi*i/100);
end
```

Sans cette opération de préallocation l'intepréteur de MATLAB "re-dimensionne" le vecteur y à chaque itération.

1.15 Entrées et Sorties: fichier I/O

Les fonctions du fichier I/0 de MATLAB sont basées sur celles du langage C. **Création d'un fichier** L'ouverture d'un fichier et la fermeture d'un fichier se font par:

```
fichier=fopen('nom', string)
       commandes
fclose('fichier'}
```

où $string$ est l'une de charactères suivants:

'r' : pour lire

'w' : pour écrire

'r+' : pour écrire et lire.

Pour **écrire** dans un fichier texte on utilise fprintf avec les formats suivants:

% e : notation exponentielle

% f : notation flottante.

% g : notation indifférenciée.

Par exemple:

```
x=-1:0.1:1; y=[x;exp(x)];
fichier=fopen('exp.txt','w');
fprintf(fichier,'Fonction exponentielle' \n');
fprintf(fichier,'\% 6.2e   \% 12.8f\n',x,y);
fclose('fichier');
```

Il y a création d'un fichier texte constitué de deux colonnes de nombres: les nombres de la première ont six chiffres dont deux chiffres aprés la virgule et ceux de la seconde ont douze chiffres dont huit aprés la virgule.

Le symbole \n signifie un passage à la ligne. Pour **lire** dans un fichier texte on utilise fscanf avec les formats suivants:

% s : lecture d'un charactère

% d : lecture d'un décimal.

% f : lecture d'un flottant.

Par exemple le fichier précédent se lit:

```
fichier=fopen('exp.txt','r');
titre=fscanf(fichier,'\s');
[x,y]=fscanf(fichier,'\d  \f');
fclose('fichier');
```

Pour écrire et lire dans des fichiers binaires on utilise fwrite et fread

1.16 Erreurs

Quand on rencontre une erreur d'exécution d'un M-fichier plusieurs méthodes sont possibles:

1- supprimer les points virgules et examiner les résultats intermédiaires.

2- utiliser le "debugger" et exécuter les commandes ligne par ligne.

Les commandes du debugger sont:

dbstop : recherche du point d'erreur.

dbclear : efface le point d'erreur.

dbcont : résume l'exécution.

dbdown : change vers le bas le contexte de l'espace de travail local.

dbstack : liste qui appelle qui.

dbstatus : liste de tous les points d'erreur.

dbstep : exécute une ou plusieurs lignes.

dbtype : liste du M-file avec numéro de ligne.

dbup : change vers le haut le contexte d'espace de travail local

dbquit : quitte le mode debug.

1.17 Polynômes

conv : multiplication polynomiale.

deconv : division polynomiale.

poly : construit polynôme avec racines données.

polyder : différenciation polynomiale.

polyfit : polynôme au sens des moindres carrés.

polyval : évaluation polynomiale.

polyvalm : évalation d'un polynôme matriciel.

residue : développement en fraction rationnelle.

roots : racines de polynômes

1.18 Interpolation

griddata : grille de données.

interp1 : interpolation 1-D.

interp2 : interpolation 2-D;

interft : interpolation 1-D avec FFT.

1.19 Méthodes numériques non-linéaires

fmin : minimisation d'une fonction d'une variable.

fmins : minimisation d'une fonction de plusieurs variables.

fplot : représentation graphique de fonction.

fzero : zéros de fonction d'une variable.

ode23 : résolution d'équation différentielle: Runge Kutta 2-3 .

ode45 : résolution d'équation différentielle: Runge Kutta 4-5.

quad : intégration par la quadrature de Simpson adaptée..

quad8 : intégration par quadrature de Newton-Cotes adaptée.

1.20 Référence:

' Matlab user's guide.

2. Résolution numérique des systèmes linéaires

2.1 Généralités

Le problème s'énonce ainsi : étant données deux matrices M et B, trouver une matrice X solution de

$$MX = B \quad \text{ou} \quad XM = B.$$

Le problème $XM = B$ est équivalent à $M'X' = B'$. La solution quand elle existe est calculée de la façon suivante ave Matlab

- X=M\ B est la solution de MX=B. (opérateur backslash \)
- X=B/M est la solution de XM=B. (opérateur slash /)

On a évidemment

M'\B'=(B/M)'

Soit M de taille $m \times n$. Les opérateurs backslash ou slash utilisent différentes méthodes pour calculer X en déterminant si

1- $m = n$: le système est carré.

2- $m < n$: le systèst sous déterminé.

3- $m > n$: Le système est surdéterminé.

D'autre part le problème $MX = B$ où A et B sont deux matrices, est équivalent à un problème du type $Ax = b$ où A est une matrice et b un vecteur. Il suffit de considérer'les q problèmes $MX_1 = B_1$,...,$MX_q = B_q$ où q est le nombre de colonnes de B. Les commandes qui permettent de calculer A et b à partir de M et B sont

```
[m,n]=size(M);
[p,q]=size(B);
A=zeros(q*m,q*n);
b=zeros(q*p,1);
for k=1:q,
    A([1:m]+m*(k-1)  , [1:n]+ n*(k-1))=M;
    b((k-1)*p+[1:p]  ,1)=B(:,k);
end
```

Le résultat suivant énonce quel est la structure d'une solution de $Ax = b$

Theorem 2.1.1 *La solution générale du système $Ax = b$ est égale à la somme d'une solution particulière de $AX = b$ et de la solution générale de $Ax = 0$.*

Résoudre $Ax = 0$ signifie calculer une base du noyau de A : ceci est réalisé par la commande null(A).

Précisons ce que fait l'opérateur backslash.

2.1.1 Système carrés non singuliers

Si A est inversible, alors A\ b est la solution du systèmeà la condition que le nombre de lignes de A et de b soient égales. Pour s'en persuader considŕons A une matrice 100×100 au hasard et b le vecteur somme des colonnes de A. De sorte que $un = ones(100, 1)$ soit solution de $Ax = b$. En calculant $||A\backslash b - ones(100, 1)||$ on vérifie l'acceptabilité du calcul numérique effectué.

```
A=randn(100,100);
b=sum(A,2);
A\b;
norm(ans-ones(100,1))
```

donne une erreur de l'ordre de 1e-14.

2.1.2 Systèmes carrès singuliers

Dans ce cas A n'est pas inversible. Deux cas sont à étudier.

Cas $b \in Im(A)$

Si le vecteur b appartient à l'image de A alors il existe x_0 tel que $Ax_0 = b$. la solution de $Ax = b$ est la somme de x_0 et d'un élément du noyau de A. Par exemple, les commandes

```
A=[1 2 3; 4 5 6; 7 8 9];
b=sum(A,2)
x0=A\b
A*x0-b
```

donne

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.541976e-018.
(Type "warning off MATLAB:nearlySingularMatrix" to suppress this warning.)
x0 =
     0
     3
     0
```

Matlab détecte la singularité de A et calcule une solution particulière. En faisant

warning off

le message annonçant que la matrice est singulière ne s'affichera pas. Par contre il est utile de vérifier que $||Ax0 - b||$ est proche de zéro ou nul. Voici un exemple. Tout d'abord on construit une matrice $m \times m$ de rang k de la forme

$$A = \begin{pmatrix} A_1 & A_2 \\ A_3 & A_3^{-1} A_1 A2 \end{pmatrix}$$

tel que la matrice A_1 soit inversible et de taille $k \times k$.

```
n=100;              k=50;
A1=randn(k,k);      A2=randn(k,n-k);
A3=randn(n-k,k);    A4=A3*A1^(-1)*A2;
A=[A1,A2;A3,A4];
det(A)
rank(A)
```

On trouve bien det(A)=0 et rank(A)=50.

Ensuite

```
b=sum(A1,2);
x0=A\b;
norm(A*x0-b)
```

on trouve que $||Ax_0 - b||$ est proche de 1e-14.

Cas $b \notin A$

Dans ce cas le système $Ax = b$ n'a pas de solution. L'opérateur backslash calcule la solution au sens des moindres carrés Par exemple

```
A=[1 2 3; 4 5 6; 7 8 9];
b=[0,0,1]
x0=A\b
difference=A*x0-b
```

donne

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.541976e-018.
(Type "warning off MATLAB:nearlySingularMatrix" to suppress this warning.)
x0 =
  1.0e+015 *
  -4.50359962737049
   9.00719925474099
  -4.50359962737050
difference =
     0
     4
     7
```

Ceci n'est bien sur pas satisfaisant. Il est préférable de calculer la solution au sens des moindres carrés en utilisant le pseudo-inverse de A.

```
x0=pinv(A)*b
A*x0-b

x0 =
   0.30555555555556
   0.05555555555556
  -0.19444444444444
difference =
  -0.16666666666667
   0.33333333333333
  -0.16666666666667
```

Maintenant avec

```
null(A)
```

on obtient une base du noyau de A

```
ans =
  -0.40824829046386
   0.81649658092773
  -0.40824829046386
```

2.1.3 Système sous-déterminés

Il y a moins d'équations que d'inconnues. On cherche une solution particulière et une base du noyau.

On donne un exemple avec affichage au format rat. Les commandes

```
A=[1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15];
b=[1,2,-1];
A\b
```

donne le rang et la solution particulière

```
Warning: Rank deficient, rank = 2  tol =   2.0770e-014.
ans =
    -2/3
      0
      0
      0
     7/15
```

Ensuite on calcule une base du noyau

```
format long
Ker=null(A)
A*Ker
```

On a bien R*Ker=0

```
Ker =
   0.57004416998708  -0.17635057423145   0.20964283730187
  -0.74547800114907   0.22353506993016   0.30707429445348
  -0.17015792915850  -0.36521356429208  -0.79853949908665
   0.29657318181590   0.76522421571948  -0.16271523439464
   0.04901857850459  -0.44719514712611   0.44453760172594
ans =
  1.0e-014 *
  -0.04440892098501                  0   0.04440892098501
  -0.39968028886506   0.08881784197001   0.08881784197001
  -0.71054273576010   0.26645352591004  -0.08881784197001
```

2.1.4 Systèmes surdéterminés

Il y a plus d'équations que d'inconnues. Si $Ax = b$ admet une solution alors elle est donnée par $A\backslash b$. Par exemple

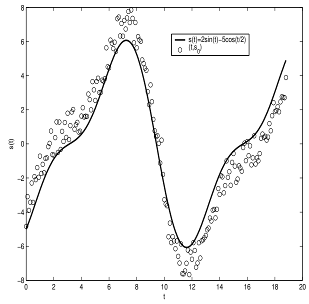
```
A=[-1, 1;1 1;3 1];  b=[1 2 3];
x0=A\b
A*x0-b
```

donne

```
x0 =
   0.50000000000000
   1.50000000000000
ans =
  1.0e-015 *
   0.66613381477509
                  0
  -0.44408920985006
```

Dans le cas le plus courant il n'y a qu'une solution au sens des moindres carrés. On donne un exemple de problème classique donnant lieu à un problème de moindres carrés linéaires. Supposons qu'un signal soit représenté exactement par la fonction $s(t) = 2 * sin(t) - 5 * cos(t)$. Pour des raisons d'erreurs dans les mesures on trouve des points $(t_i, s(t_i))$ comme le montre le graphique ci dessous. Ce dernier a été obtenu en perturbant les vraies valeurs par les commandes suivantes

```
t=[0:0.1:6*pi]';     s=2.*sin(t)-5*cos(t/2);
s0=s+2*[rand(100,1),-rand(89,1)];
clf,  plot(t,s,'k-','linewidth',2),  hold on,  plot(t,s0,'ko')
```



On veut calculer une courbe de la forme $s(t) = a_1 sin(t)(t) + a_2 cos(t/2)$ qui reproduise le signal perturbé. Pour cela on résoud au sens des moindres carrés le système linéaire

acos(t)+bsin(t)=s0

avec t=[0:0.1:6*pi] La solution est donnée par

```
a=[cos(t),sin(t)]\s0;
```

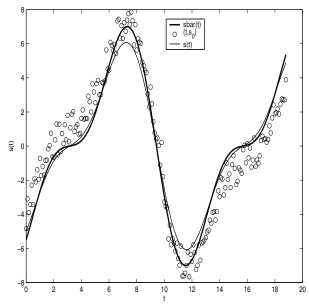
On trouve

```
a =
   2.59733645497733
  -5.49051533394099
```

Puis par les commandes ci-dessous

```
sbar=a(1)*sin(t)+a(2)*cos(t/2);
clf,plot(t,sbar,'k-','linewidth',2)
hold on,plot(t,s0,'ko',t,s,'k-')
```

on obtient le graphique:



2.2 Méthodes directes

2.2.1 Systèmes triangulaires

On note par U une matrice triangulaire supérieure et par L une matrice triangulaire inférieure. Les deux fonctions Matlab tril(L,b) et triu(U,b) résolvent $Lx = b$ et $Ux = b$.

Voici une version Matlab de ces fonctions. La fonction $triL$ résoud les systèmes triangulaies inférieurs. Cet algorithme repose sur l'écriture

$$\begin{pmatrix} l_{11} & 0 \\ l_{2:n} & L_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_{2:n} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_{2:n} \end{pmatrix}$$

Il vient $x_1 = \dfrac{b_1}{l_{11}}$ et $L_{22}x_{2:n} = b_{2:n} - x_1 l_{2:n}$.

```
function x=triL(L,b,n)
x=b;
for k=1:n
    x(k)=x(k)/L(k,k);
    x(k+1:n)= x(k+1:n)-x(k)*L(k+1:n,k);
end
```

La fonction $triU$ résoud les systèmes triangulaies supérieurs. Cet algorithme repose sur l'écriture

$$\begin{pmatrix} U_{n-1n-1} & u_{1:n-1} \\ 0 & u_{nn} \end{pmatrix} \begin{pmatrix} x_{1:n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_{1:n-1} \\ b_n \end{pmatrix}$$

Il vient $x_n = \dfrac{b_n}{u_{nn}}$ et $U_{n-1n-1}x_{1:n-1} = b_{1:n-1} - x_n u_{1:n-1}$.

```
function x=triU_direct(U,b,n)
x=b;
for k=n:-1:1
    x(k)=x(k)/U(k,k);
    x(1:k-1)= x(1:k-1)-x(k)*U(1:k-1,k);
end
```

2.2.2 Méthode(s) de Gauss

Soit $Ax = b$ un système linéaire avec A une matrice de taille $n \times n$ inversible. On sait qu'il existe dans ce cas une unique factorisation LU de A :

$$PAQ = LU$$

où P et Q sont des matrices de permutation, L une matrice triangulaire inférieure avec une diagonale formée de 1 et U une matrice triangulaire supérieure. L'algorithme de Gauss consiste à calculer L et U. Ensuite il suffit de résoudre successivement les deux systèmes triangulaires :

1- $Ly = PbQ$

2- $Ux = y$ Sous matlab pour résoudre un système $Ax = b$ il suffit de faire

$$\mathrm{x} = A^\backslash b$$

Voici à titre d'exemple une implémentation de la factorisation LU d'une matrice de taille $m \times n$. On donne le choix d'une méthode avec pivot total ou partiel ainsi que d'afficher l'évolution de la matrice A au cours des itérations. Il faut noter que la partie inférieure de la matrice A sert à stocker la matrice L tandis que la partie supérieure de A stocke la matrice U. Plus intéressant la fonction ci-dessous offre la possibilité de calcul du facteur de croissance ρ qui caractérise le rapport du pivot maximun sur la norme infinie de la matrice A

$$\frac{\max_{i,j,k}(|a_{ij}^{(}k)|)}{\max_{i,j}|a_{i,j}|}$$

```
function [L,U,P,Q,rho]=lupq(A,pivot,affiche)
%LUPQ   factorisation lu avec pivot partiel ou total
%       [L,U,P,Q]=lupq(A,pivot) calcule une factorisation P*A*Q=L*U
%       de la matrice A de taille m par n avec m>=n.
%INPUTS
%       A= matrice m par n
%       pivot definit la methode :
%           - pivot='t' : pivot total
%           - pivot='p' : pivot partiel
%       affiche = optionnel. Affiche la matrice A a chaque etape si
%       affiche=1
%
%OUTPUTS
%       L=matrice triangulaire inferieure m par n avec des 1 sur la diagonale
%       U=matrice triangulaire superieure n par n
%       P=matrice de permutation m par m
%       Q=matrice de permutation n par n
%       rho=facteur de croissance
%           = max_{i,j,k}( a_{ij}^(k) )/ max_{i,j} a_{i,j}
%
%       Par defaut lupq(A) applique la methode du pivot total
%       A chaque etape de la methode de factorisation on a
%            L(i,j) =A(i,j) pour i>j
%            U(i,j)=A(i,j) pour i<=j
%
[m,n]=size(A);
maxA=norm(A,'inf'); rho=maxA;
if m<n, error('La matrice initiale doit avoir m_Lignes>= n_Colonnes'),end
if nargin<2 or pivot ~='p', pivot='t';end
p=1:m; q=1:n;
for k=1:min(m-1,n)
    if pivot=='t'
        %calcul de  l'indice de ligne et de colonne du max de |A(k:m,k:n)|
        [colmax,iligne]=max(abs(A(k:m,k:n)));
        [cmax,icol]=max(colmax);
        ligne=iligne(icol)+k-1;
        col=icol+k-1;
    elseif pivot=='p'
```

```
        %calcul de l'indice de ligne du max de la colonne |A(k:m,k)|
        [colmax,iligne]=max(abs(A(k:m,k)));
        ligne=iligne(1)+k-1;        col=k;
    end
   % permutation des lignes et colonnes de A et mise a jour de p et q
    A([k,ligne],:)=A([ligne,k],:);    A(:,[k,col]) =A(:,[col,k]);
    p([k,ligne])=p([ligne,k]);         q([k,col])=q([col,k]);
    if A(k,k)==0
        if pivot=='t', break,end
    else
        ik=k+1:m;
        A(ik,k)=A(ik,k)/A(k,k);
       if k+1<=n
            jk=k+1:n;
            A(ik,jk)=A(ik,jk)-A(ik,k)*A(k,jk);
            if nargout>=5, rho= max(rho, max(max(abs(A(ik,jk)))));end
        end
    end
   if nargin>=3 && affiche==1, A,end;
end
if nargout<=1, L=A;return,end
%L,U
L=tril(A,-1)+eye(m,n);
U=triu(A);
U=U(1:n,:);
%P,Q,rho
if nargout >= 3, P = eye(m); P = P(p,:); end
if nargout >= 4, Q = eye(n); Q = Q(:,q); end
if nargout >= 5, rho = rho/maxA; end
```

2.2.3 Étude numérique du facteur de croissance

L'analyse d'erreur rétrograde de l'erreur de l'algorithme du pivot de Gauss fait intervenir le facteur de croissance

$$g_n(A) = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}.$$

Wilknson montre que si l'algorithme de Gauss produit une solution approchée $\hat{x}$ de $Ax = b$ tel que $(A + \Delta A)\hat{x} = b$ alors

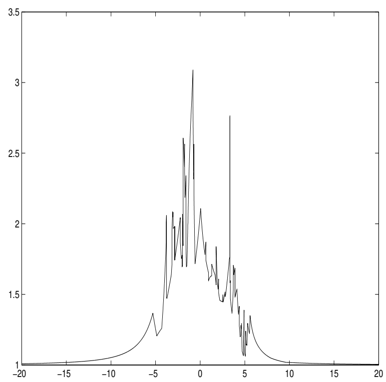
$$||\Delta A||_\infty \leq 2n^2 \gamma_m g_n(A) ||A||_\infty,$$

avec $\gamma_n = \dfrac{nu}{1 - nu}$ et u l'unité d'arrondi. (Sous Matlab on a $u = eps/2$ avec eps $=$ 2.220446049250313e $-$ 016.) Sous ce point de vue, il est intéressant d'avoir une borne supérieure de $g_n(A)$. Wilkinson a montré que $g_n(A)$ est borné par

$$g_n(A) \le \sqrt{n}\left(2\,3^{1/2}\,\ldots n^{\frac{1}{n-1}}\right)^{1/2}$$

pour l'algorithme de Gauss avec pivot total. En pratique cette borne est beaucoup trop grande. Pour illustrer ce fait on considère une matrice au hasard perturbé par eId avec $-20 \le e \le 20$. Par les commandes

```
A=randn(20,20);
a=20;   r=[];
for e=-a:1/1000:a,
   B=A+diag(e.*ones(1,20));
   [L,U,P,Q,rho]=lupq(B,'p');
   r=[r,rho];
   clf,
   plot(-a:1/1000:a,r,'r-')
end
```



$g(A+eId)$ pour $-20 \le e \le 20$.

on obtient le graphique ci-dessus.

3. HOWTO

On donne quelques

3.1 Graphique

3.1.1 Sauver un graphique et l'inclure dans un texte LATEX

Produire le graphique et le sauver à l'aide du menu file et export. Une fenêtre s'ouvre. Choisir le répertoire de sauvegarde et sauver le fichier sous la forme

nom_de_fichier.eps

Dans le fichier source LATEX faire

```
\begin{figure}
\includegraphics[width=4cm,height=4cm]{fichier.eps}
\caption{titre}
```

4. Analyse Matricielle

4.1 Conditionnement de la résolution des systèmes linéaires

Soit $f\,:\,(A,b,x)\in\mathcal{M}_{n\times n}(\mathbb{C})\times\mathbb{C}^n\times\mathbb{C}^n\to Ax-b\in\mathbb{C}^n$.

Definition 4.1.1 *Le conditionnement d'une matrice est la quantité*

$$K(A)=||A||\,||A^{-1}||.$$

On va voir que le conditionnement d'une matrice est relié à l'erreur relative d'un problème de résolution d'un système linéire.

Theorem 4.1.2 *Supposons A inversible et considérons une matrice M telle que $||M-A||<\dfrac{1}{||A^{-1}||}$.*

1- M est inversible et $||M^{-1}A||\leq\dfrac{1}{1-||A^{-1}||\,||M-A||}$.

2- Soient (A,b,x) et (M,p,y) tels que $Ax-b=My-p=0$. Alors

$$\frac{||y-x||}{||x||}\leq\frac{K(A)}{1-K(A)\frac{||M-A||}{||A||}}\left(K(A)\frac{||M-A||}{||A||}+\frac{||p-b||}{||b||}\right).$$

Preuve. La dérivée $D_xf(A,b,x)$ de la fonction $f(A,b,x)=Ax-b$ est égale à A. Si A est inversible, il existe une fonction implicite telle que $f(A,b,\varphi(A,b))=0$ définie dans un voisinage de (A,b). Montrons que M est inversible pour tout M vérifiant $||M-A||<||A^{-1}||^{-1}$. On a $A^{-1}M=I+A^{-1}(M-A)$.Donc $||A^{-1}M||\geq 1-||A^{-1}||\,||M-A||>0$. On en déduit le 1.

Majorons l'erreur relative. Puisque $My-p=Ax-b=0$ on a

$$\begin{aligned}0=My-p&=(A+M-A)(x+y-x)-b-(p-b)\\&=Ax-b\quad+(M-A)x-(p-b)+A(y-x)\quad+(M-A)(y-x)\\&=(M-A)x-(p-b)+M(y-x).\end{aligned}$$

Donc $y - x = M^{-1}A\,A^{-1}\left((M-a)x - (p-b)\right)$. D'autre part $Ax = b$ et $||A||\,||x|| \leq ||b||$. On a donc

$$\begin{aligned}
\frac{||y-x||}{||x||} &\leq ||M^{-1}A|| \quad ||A^{-1}||\,||A|| \left(||M-A||\,||A^{-1}|| + \frac{||p-b||}{||b||}\right) \\
&\leq \frac{K(A)}{1 - K(A)\frac{||M-A||}{||A||}} \left(K(A)\frac{||M-A||}{||A||} + \frac{||p-b||}{||b||}\right).
\end{aligned}$$

□