Introduction à MATLAB

PRESENTATION GENERALE (1)

- MATLAB est la contraction de « Matrix Laboratory », et est devenu depuis quelques années un des outils de choix des scientifiques pour l 'analyse des signaux et images.

- Comme ceci l'indique, MATLAB a d 'abord été créé pour faciliter les calculs matriciels. Mais de nombreuses fonctionnalités mathématiques et graphiques ont été ensuite rajoutées.

- MATLAB:
  - permet de travailler indifféremment sur les nombres réels et complexes.
  - Permet de travailler directement sur les vecteurs et matrices.
  - Contient un nombre de commandes impressionnant pour la plupart des opérations mathématiques (fonctions …).
  - Permet de réaliser tout un assortiment de sortie graphiques.
  - Permet à l ’aide d ’une programmation simple de créer de nouvelles commandes.

- MATLAB propose également un grand nombres de « toolboxes » pour:
  - le traitement des signaux
  - le traitement des images
  - les statistiques
  - les réseaux de neurones
  - la logique floue
  - la simulation de systèmes
  - ...

- Quelques commandes de base vraiment utiles:

who	liste les variables existantes
whos	pareil avec les détails
help *commande*	décrit ce que fait *commande*
lookfor *mot-clé*	liste les commandes dont la description contient *mot-clé*

- Comparez ce qui se passe (avec `who`)quand vous tapez:

  `sqrt(2)`   `a = sqrt(2)`   `b = sqrt(2);`

- Nous allons maintenant calculer les racines de:

  $ax^2 + bx + c = 0$ dont les racines sont: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- Tapez:

```
a = 1          b = 0          c = 2
x1 = (-b + sqrt(b^2 - 4*a*c) / (2*a)
x2 = (-b - sqrt(b^2 - 4*a*c) / (2*a)
```

- Refaites l 'expérience avec:

```
a = 1          b = 1          c = 1
```

COMMANDES DE BASE (3)

- Tapez

  x=1:20   y=1:2:20   z=0:0.1:3   x(11)   x(3:7)

- Tapez

  y = sin(x)   plot(y)

- Tapez

  y = x. * x   plot(y)

- Tapez

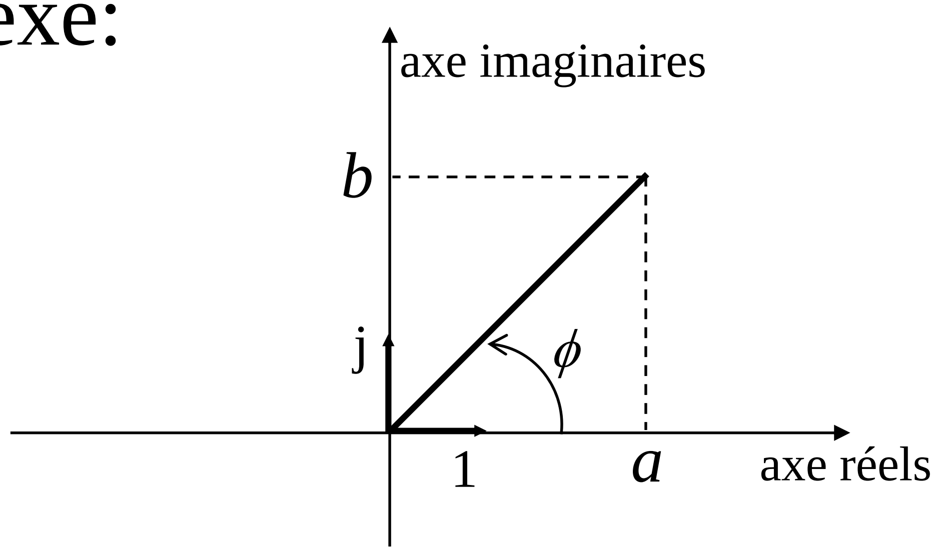
  x = zeros(3,3)   x = ones(3,3)   x = [1 5 ; 7 9]

- Tapez

  x = zeros(3,1)   x = ones(1,3)   x = [1 5 7 9]   x = [1 5 7 9] ’

  x = 5+3*j   conj(x)   abs(x)

- Définition succincte: un nombre complexe a la forme $z = a + \mathrm{j}b$, a est la partie réelle de z, b la partie imaginaire, et j est tel que $\mathrm{j}^2 = -1$.
- Il est pratique de représenter ces nombres dans le plan complexe:



- Définitions et propriétés:
  - conjugué: $z^* = a - \mathrm{j}b$
  - module: $|z| = \sqrt{a^2 + b^2}$
  - rep. polaire: $z = |z| (\cos \phi + \mathrm{j} \sin \phi)$
  - somme: $(a + \mathrm{j}b) + (c + \mathrm{j}d) = (a + c) + \mathrm{j}(b + d)$
  - produit: $(a + \mathrm{j}b)(c + \mathrm{j}d) = (ac - bd) + \mathrm{j}(ad + bc)$
  - on a: $|z^*| = |z|$ et $zz^* = |z|^2$ et $(z_1 z_2)^* = z_1^* z_2^*$
    et $|z_1 z_2| = |z_1||z_2|$
  - d 'où quotient: $\dfrac{z_1}{z_2} = \dfrac{z_1 z_2^*}{z_2 z_2^*} = \dfrac{z_1 z_2^*}{|z_2|^2}$

- Formule d'Euler et ses conséquences
  - formule : $e^{j\phi} = \cos\phi + j\sin\phi$ (écrit aussi $\exp(j\phi)$)
  - d 'où: $z = |z| \exp(j\phi)$
  - on a: $|\exp(j\phi)| = \sqrt{[\cos(j\phi)]^2 + [\sin(j\phi)]^2} = 1$
  - on a aussi: $[\exp(j\phi)]^* = \cos\phi - j\sin\phi$

    $= \cos -\phi + j\sin -\phi = \exp(-j\phi)$
  - donc: $\exp(j\phi)\,[\exp(j\phi)]^* = |\exp(j\phi)|^2 = 1$

    $[\exp(j\phi)]^* = \exp(-j\phi) = 1/\exp(j\phi)$
  - et: $z_1 z_2 = |z_1|\exp(j\phi_1).\,|z_2|\exp(j\phi_2) = |z_1|\,|z_2| \exp[j(\phi+\phi_2)]$

- A l'aide des commandes abs, angle, conj, real, imag, et exp, vérifiez les propriétés sur somme, produit, module du conjugué, conjugué et module d 'un produit, ainsi que celles provenant de la formule d'Euler en prenant des nombres complexes quelconques. Visualisez-les aussi dans le plan complexe avec compass (et hold).
- Visualisez avec compass la position dans le plan complexe de nombres de la forme:

$$z(k) = \exp(j2\pi f k), \quad f = 0.1, \; k = 0, \ldots, 9$$

SORTIES GRAPHIQUES (1)

- Il y a tant de possibilités graphiques en Matlab que nous n'allons explorer que les plus courantes.
- Tapez les commandes suivantes:

```
x = -10:0.4:10;          y = sin(0.2*pi*x);
plot(y)    plot(x,y)    plot(x,y, 'g ')        plot(x,y, '+ ')
plot(x,y, 'g ',x,y, '*m ')        stem(y)      stem(x,y)
semilogx(y)              title(' graphe ')     xlabel(' log x ')
axis(' square ')         axis(' normal ')
subplot(211)   plot(x,y)     subplot(212)         plot(x,y.*y)
axis([-20 20 -3 3])
```

SORTIES GRAPHIQUES (2)


```
bar(y)            bar(x,y)           bar(x,y, 'g ')
z = randn(5000,1);                   hist(z,30)
z = rand(5000,1);           hist(z,30)
t = .01 : .005 : 0.99;      x = cos(20*pi*t);      y =
sin(20*pi*t);
 plot(t,x,'r'); hold on; plot(t,y,'r'); hold off
 gtext('sinus et cosinus')
zoom
load trees      imshow(X,map)
```

- On peut également visualiser des fonctions bi-dimensionnelles. Avec des courbes de contour:

```
[X,Y] = meshgrid(-3:1/8:3);
Z = peaks(X,Y).*sin(X);
v1 = -4:-1;
v2 = 0:4;
contour(Z,v1,'k- -');
hold on
contour(Z,v2,'k-');
hold off
```

- Passons maintenant à de vraies représentations 3D. Regénérez les données:

```
[X,Y] = meshgrid(-3:1/8:3); Z = peaks(X,Y);
```

- Et essayez:

```
plot3  mesh  meshc
surf   shading   colormap
v1 = -4:-1; C=contour(Z,v1,’k--’); clabel(C,v1)
```

- Représentez en 3D les valeurs (incrément de 0.1) de la fonction:

$$z = \sin(x).\ \sin(y) \text{ pour } x,y \in [-\pi, \pi]$$

- Même chose pour:

$$z = x - 0.5x^3 + 0.2y^2 + 1 \text{ pour } x,y \in [-3, 3]$$

- Observez ce que fait la commande clabel

- Générez les commandes:

A=randn(3,3)   A(:,1) A(2,:) A(1:2, 2:3) A’

- Un peu d’algèbre linéaire:

A=A’*A   inv(A) inv(A)*A det(A)

[V,D]=eig(A)   A*V(:,1)   D(1,1)*V(:,1)

[B,C]=qr(A)   B*C

- Un peu de magie:

A=magic(5)   sum(A)   sum(A’)   trace(A)

- Régression (approximation) linéaire. Tapez les commandes:

```
x=0:9   y=2*x+randn(1,10)  P=polyfit(x,y,1)
xx = 0:0.1:9    yy=polyval(P,xx)
plot(x,y,'*r')      hold on   plot(xx,yy,'--g')  hold off
```

- Approximation polynômiale. Tapez les commandes:

```
x=-2:0.5:4   y=cos(2*x)    P=polyfit(x,y,5)
xx = -2:0.02:4    yy=polyval(P,xx)
plot(x,y,'db')      hold on   plot(xx,yy,'--r')  hold off
```

- Pour éviter de rentrer une série de commandes plusieurs fois, on peut toujours les regrouper dans un fichier texte *nom*.m, et taper *nom* dans Matlab pour les exécuter. On a ce qu'on appelle un script, mais pas une vraie fonction.
- Pour créer une vraie fonction, qui aura exactement les mêmes caractéristiques qu'une commande Matlab, il faut également créer un fichier avec l 'extension .m. Supposons que nous décidions d 'appeler la fonction fonc.

CREATION DE FONCTIONS (2)

- Il faut tout d 'abord créer un fichier texte fonc.m.
- Le texte dans fonc.m devra toujours avoir la forme:

  function [out1, …, outn] = fonc(in1, …, inm)

  % commentaires de début

  Calculs
- Les paramètres d'entrée et de sortie peuvent être de tout type (scalaire, vecteur, matrice, …)
- Si on tape ensuite dans Matlab help fonc, le texte des commentaires de début apparaît.
- A part si on veut afficher un résultat intermédiaire, on met en général un ; à la fin de chaque commande de *Calculs*.

- Créez un fonction racines2 donnant les racines d 'un polynôme d 'ordre 2 à partir des coefficients de ce polynôme.
- Créez un fonction racines3 donnant les racines d 'un polynôme d 'ordre 3 à partir des coefficients de ce polynôme, en utilisant la commande Matlab roots.
- Créez une fonction puis3 calculant la puissance 3 d 'un ensemble de nombres contenu dans un vecteur. Essayez cette fonction avec x = -20:20 et plot.

- Dans la partie *Calculs* d'une fonction, il est possible d 'effectuer des opérations répétitives avec une boucle for. La structure générale est:

```
for k=n:m,
    commande(s)
end
```

- Il est également possible d 'exécuter des commandes conditionnellement avec un if:

```
if condition
    commande(s)
else
    autre(s) commande(s)
end
```

- Créez une fonction sinv calculant le sinus des composantes d'un vecteur à l'aide d'une boucle for et de la commande length. Comparez le temps d 'exécution de celle-ci avec celui de la commande directe sin pour un vecteur de 10000 composantes. Conclusion?
- Créez une fonction absv (entrée x, sortie y) telle que la $i^{\text{ème}}$ composante de y est 1 si la $i^{\text{ème}}$ composante de x est positive, et 0 sinon, à l'aide de for et if. Comparez le temps d 'exécution de celle-ci avec celui de la commande directe x > 0 pour un vecteur de 10000 composantes. Conclusion?