Sup Galilée 2021–2022
MACS1

Rappel des commandes Octave importantes

0.1 Aide dans Octave

Pour obtenir de l'aide dans Octave sur une commande, on peut
- taper `help macommande` dans la fenêtre de commande
- chercher dans la documentation graphique (onglet sous la zone "Editeur")

0.2 Raccourcis clavier importants (peuvent dépendre du système)

F5 : sauver et exécuter ctrl-S : sauver alt-2 : changer de fenêtre Octave

1 Avant toute chose

`clear all;close all;`

Au début d'un script, cette ligne permet d'effacer toutes les variables et de fermer toutes les fenêtres graphiques. Cela peut éviter bien des erreurs.

2 Graphisme élémentaire

2.1 Gestion des fenêtres graphiques

`figure(k)` **affiche** la fenêtre graphique `k`, et en fait la fenêtre courante (tous les tracés se feront dans cette fenêtre)

`clf` **efface** le contenu de la fenêtre courante

`hold on` permet de demander à ce que les graphes affichés dans la même fenêtre se **superposent** au lieu de se remplacer. Exemple : `figure(1);clf;hold on; plot(...);plot(...);plot(...);`

`subplot(m,n,k)` découpe la fenêtre courante en `m` lignes et `n` colonnes, et fait de la **sous-fenêtre** numéro `k` (dans l'ordre de lecture) la fenêtre courante. Ex. avec 2 graphes : `subplot(2,1,1);plot(...);subplot(2,1,2);plot(...);`

2.2 Commande `plot`

Si `x` et `y` sont des **vecteurs lignes** de même taille, `[x1 ··· xn]` et `[y1 ··· yn]`, alors `plot(x,y)` représente graphiquement les points du plan (`x1`,`y1`),...,(`xn`,`yn`) et les relie par des segments de droite.

Si `x` et `y` sont des **matrices** de même taille, alors `plot(x,y)` représente, sur la même fenêtre graphique, les courbes correspondant à chaque ligne de `x` et `y`.

Choix des paramètres d'affichage.
- Version abrégée : `plot(x,y,'.-r')` représente les points par des petits disques (`.`), les segments en trait continu (`-`) et le tout en rouge (`r`). On peut remplacer `.` par `*`,`+`,`o` (ou par rien du tout). On peut remplacer `-` par `:`, `--`, `-.`. On peut remplacer `r` par `y`,`m`,`c`,`r`,`g`,`b`,`w`,`k`. Avec plusieurs graphes : `plot(x1,y1,'-r',x2,y2,'.b')` et on peut spécifier alors une légende entre `";"` : `plot(x1,y1,'-r;graphe1;',x2,y2,'.b;graphe 2;')`
- Version complète : `plot(x,y,'Paramètre1',Valeur1,'Paramètre2',Valeur2,...)` permet de choisir tous les paramètres, par exemple

Paramètre	Signification	Exemples de valeurs
`Color`	couleur du trait	`'red','r'`
`Marker`	type de marqueur de point	`'.','+'`
`MarkerEdgeColor`	couleur des traits des marqueurs	`'blue','b'`
`MarkerFaceColor`	couleur de remplissage des marqueurs	`'black','k'`
`MarkerSize`	taille des marqueurs	`5.5`
`LineStyle`	type de trait	`'-','-.',':','--'`
`LineWidth`	épaisseur du trait	`2.5`

2.3 Choix des axes

Lors de l'appel à `plot`, Octave choisit des axes adaptés à la figure. Il peut être ensuite utile de les modifier.

- `axis equal` force les axes à être **orthonormés**.
- `xlim([xmin xmax])` force l'axe des abscisses à aller de `xmin` à `xmax`
- `ylim([ymin ymax])` force l'axe des ordonnées à aller de `ymin` à `ymax`

2.4 Titre et légendes

La commande `title('Titre du graphique')` permet de donner un **titre** (remarque : pour écrire une apostrophe dans un titre, il faut taper 2 apostrophes : `'Titre d''un graphe'`). On peut préciser des paramètres comme avec `plot` : `title('Titre du graphique','Paramètre1',Valeur1,...)`. Par exemple :

Paramètre	Signification	Exemples de valeurs
`Color`	couleur du texte	`'green'`, `'g'`
`FontSize`	taille du texte	`10`

Les commandes `xlabel('Nom abscisse')` et `ylabel('Nom ordonnée')` permettent d'indiquer le **nom des axes**. On peut préciser des paramètres comme pour le titre.

Si plusieurs courbes apparaissent sur le même graphe, on peut les distinguer en affichant une **légende** : `legend('Courbe 1','Courbe 2',...)` affiche une légende avec les noms `Courbe 1`, `Courbe 2`, etc. pour les différentes courbes par ordre de tracé.

2.5 Graphes à échelle logarithmique

De la même façon que `plot`, on peut utiliser `semilogx` pour que l'axe des abscisses suive une échelle logarithmique, `semilogy` de même pour les ordonnées, et `loglog` pour que les deux axes suivent des échelles logarithmiques.

NB. Avec `semilogx` ou `loglog`, pour que les abscisses des points représentés soient régulièrement réparties, il faut donc les répartir exponentiellement : par exemple, `x=10.^linspace(0,15,100)` pour 100 valeurs entre 1 et 10^{15}

2.6 Graphes en 3D

Si `x,y,z` sont des vecteurs lignes, `plot3(x,y,z)` fonctionne comme `plot` mais en 3D. Cela permet d'afficher des ensembles de **points**, ou des **courbes** paramétrées dans $\mathbb{R}^3$

Si `x,y` sont des vecteurs lignes de taille m et n, et `z` est une matrice de taille $n \times m$, alors `surf(x,y,z)` représente la **surface** paramétrée formée par les points `(x(i),y(j),z(j,i))` (avec une couleur qui dépend de `z(j,i)`).

2.7 Histogrammes

Si `v` est un vecteur, on peut représenter un histogramme des valeurs de ses composantes de 2 façons :

- `hist(v,k)` découpe la plage des valeurs des composantes en `k` classes (intervalles), et représente le nombre de composantes appartenant à chacune
- `hist(v,centres)` de même, avec des classes dont les centres sont donnés par le vecteur `centres`.

De plus, `hist(v,k ou centres,s)` normalise l'histogramme de façon à rendre la somme des colonnes égale à `s`.
Pour davantage de flexibilité, on décompose souvent plutôt en 2 étapes : de façon équivalente, on peut appeler

1. `[nelements,centres]=hist(v,k);` (k est le nombre de classes souhaité, ou un vecteur donnant les centres des classes) Cette commande ne dessine rien, et renvoie le vecteur des nombres d'éléments par classe et le centre des classes. Puis
2. `bar(centres,nelements,1);` (le dernier paramètre est la largeur relative des barres : 1 donne des barres adjacentes) Cette commande affiche l'histogramme.

Si l'histogramme sert à approcher une **probabilité discrète**, on choisira les centres égaux aux valeurs attendues, et on normalisera l'axe des ordonnées de façon à ce que la somme des valeurs soit 1 (c'est donc une probabilité discrète), et pour cela on adaptera le code suivant :

```
centres=      ; (vecteurs des valeurs attendues, par exemple min(v):max(v) si les valeurs sont entières)
nelements=hist(v,centres);
bar(centres,nelements/length(v),1);
```

Si l'histogramme sert à approcher une **densité de probabilité**, on choisira un nombre de classes de l'ordre de $\sqrt{N}$ (pour N données) et on normalisera l'axe des ordonnées de façon à ce que la somme des aires des barres vale 1 (c'est donc une densité de probabilité), et pour cela on utilisera plutôt le code suivant :

```
[nelements,centres]=hist(v,k); (k est le nombre de classes souhaité)
h=centres(2)-centres(1); (h est la largeur de chaque classe)
bar(centres,nelements/(length(v)*h),1);
```

3 Nombres pseudo-aléatoires

3.1 Fonction `rand`

`rand` renvoie, à chaque appel, une réalisation d'une variable aléatoire indépendante des précédentes et qui suit la loi uniforme sur $[0,1]$.

Plus généralement, `rand(m,n)` renvoie une matrice de taille `[m n]` dont les coefficients sont générés selon la loi uniforme sur $[0,1]$.

3.2 Réinitialiser le générateur

`rand('seed',k)` permet de réinitialiser le générateur de nombres aléatoires avec une graine entière `k`. Ainsi, après `rand('seed',1)`, les appels à `rand` renverront la **même** suite de nombres réels. Ceci permet de reproduire à l'identique une simulation.

`rand('seed','reset')` procède de même, avec une graine qui dépend de l'heure actuelle, en millisecondes. En l'exécutant une fois au début de la session, cela permet d'avoir une suite *a priori* différente à chaque fois. *Attention*, il ne faut pas réinitialiser le générateur trop souvent, sous peine de risquer d'introduire des corrélations entre les graines et donc entre les tirages `rand`.

3.3 Variantes : autres lois

On essaiera en général de n'utiliser que `rand`. Cependant, on pourra aussi utiliser les fonctions suivantes :

- `randi(N)` renvoie une réalisation d'une variable aléatoire de loi uniforme sur $\{1,\dots,N\}$.
  Plus généralement, `randi(N,m,n)` renvoie une matrice de taille `[m n]` de tels nombres.
- `randn` renvoie une réalisation d'une variable aléatoire de loi normale standard $\mathcal{N}(0,1)$.
  Plus généralement, `randn(m,n)` renvoie une matrice de taille `[m n]` de tels nombres.

4 Calculs matriciels et vectoriels

4.1 Opérations

Opérations composante par composante : `A+B`, `A.*B`, `A./B`, `A.^2`

Opérations matricielles : `A+B`, `A*B`, `A^2`, `A^-1`, `[A B]` (juxtaposition), `[A;B]` (idem, verticalement), `flip(A)` (miroir de chaque colonne), `sort(A)` (tri de chaque colonne)

4.2 Transposée, transconjuguée, norme

`A'` renvoie la conjuguée (complexe) de la transposée de `A`

`A.'` renvoie la transposée de `A`. Si `A` est réelle, c'est équivalent à `A'`

Ainsi, si `x` et `y` sont des vecteurs colonnes, `x'*y` renvoie leur **produit scalaire** canonique (dans les cas réel et complexe)

Si `x` est un vecteur, `norm(x)` renvoie la norme euclidienne de `x`

4.3 Extraction triangulaire

`tril(A)` renvoie une matrice triangulaire inférieure, remplie à partir des coefficients correspondants de `A`.

`triu(A)` fait de même de façon triangulaire supérieure.

4.4 Spectre

`eig(A)` renvoie dans un vecteur colonne le spectre de A. Pour obtenir également les vecteurs propres, on demandera `[val vec]=eig(A)`, auquel cas `val` est une matrice diagonale donnant les valeurs propres, et `vec` une matrice carrée dont les colonnes donnent des vecteur propres.

4.5 Systèmes linéaires

`A\b` renvoie une solution du système `Ax=b`.