

TP 3 : affichage de données – partie 2

Exercice 1 : Calcul et affichage de racines d'une fonction du second degré

Créer une fonction `affiche_racine(poly)` qui permet d'afficher un polynôme du second degré `poly` et ses racines si elles existent.

La fonction du second degré `poly` sera codée sous forme d'un tableau de ses coefficients.

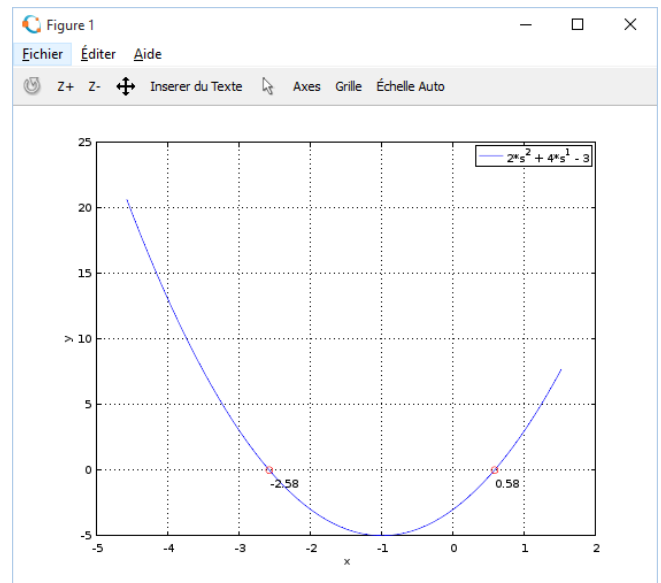
Par exemple, la fonction $y=2x^2+4x-3$ sera codée `[2 4 -3]`

L'appel à la fonction depuis un script sera donc :

```
affiche_racine([a b c])
```

`a` correspond au 1^{er} élément du tableau `poly` soit `poly(1)`, etc. ...

Les racines, si elles existent, seront affichées par un rond rouge avec leur abscisse (utiliser les fonctions `sprintf` et `text`)



Exercice 2. La fonction fplot

Une autre fonction « utile » est possible pour afficher les courbes ; la fonction `fplot` (fonction **plot**)

En donnant de manière formelle la fonction et l'intervalle d'affichage, vous pourrez tracer directement la courbe.

Tracer la fonction x^2-2 en noir dans l'intervalle `[-2 2]` en une seule ligne de code.

Nota : la variable utilisée doit nécessairement s'appeler 'x'

Élément de Cours : Afficher des courbes en 3D

Affichage 3D

Enfin, Octave via GNUpot permet d'afficher des graphes en 3 dimensions. Là encore, plusieurs commandes sont disponibles.

- `plot3` : l'équivalent en 3D de la commande `plot` prend `(x, y, z)` en arguments. Avec `plot3`, on peut ajouter `zlabel` qui fonctionne comme `xlabel` et `ylabel`

Il est possible de créer une grille de points dans lequel une fonction $z=f(x, y)$ sera affichée en utilisant les commandes `surf` (pour tracer une surface paramétrée d'équations) ou `mesh`

Il est aussi possible d'interagir directement avec les données sur le graphe à l'aide de touches ou d'actions avec la souris.

Exercice 3 : Une courbe en 3D

Afficher dans l'intervalle `[0, 30]` la fonction $f(x, y, z)$ définie par :

$$\begin{cases} x = t \\ y = \sin(t) \\ z = \cos(t) \end{cases}$$

Exercice 4 : Une surface en 3D

Définir la fonction `z = formeetrange(x, y)` qui calcule la fonction suivante

$$z = \cos(3 \cdot \sqrt[3]{x^2 + y^2})$$

Définir une fonction `affichage(bornes)` où `bornes` contient un vecteur des bornes `min/max` pour l'axe `x` et `y` qui affiche le résultat de l'équation précédente sous forme de surface (fonction `mesh`)

Afficher la fonction sur l'intervalle `[0 1, 0 1]`, `[0 2, 0 2]` et `[0 5, 0 5]`

Manipuler la forme générée avec les boutons de la souris.

Nota : Pensez à générer avant l'affichage l'ensemble des points de maillage en utilisant la fonction `meshgrid`

```
x = [0:0.1:1]
y = [0:0.1:1]
[X,Y] = meshgrid(x,y);
...
```

Exercice 5 : Des élections

Soit le résultat d'une élection mettant aux prises 4 candidats.

- Le candidat 1 (Louis) obtient : 231 voix
- Le candidat 2 (Jeanne) obtient : 424 voix
- Le candidat 3 (Maurice) obtient : 489 voix
- Le candidat 4 (Albertine) obtient : 12 voix

Le nombre total d'inscrits à l'élection est de **1412**.

Afficher les résultats des élections sous forme de barres (`bar` ou `barh`) et sous forme de camembert (`pie` ou `pie3`)

Exercice 6 : Des valeurs en série

Nous voulons afficher les valeurs discrètes de la suite de Fibonacci dans l'intervalle `[1, 16]`

1. écrire la fonction `Fibonacci(n)` qui permet de renvoyer un vecteur des `n` valeurs successives de la suite de Fibonacci
2. afficher les 16 premières valeurs de manière discrète (fonction `stem`)
3. créer le fichier `fibonacci16.jpg` correspondant au graphique du résultat

Nota : Suite de Fibonacci

$$F_1 = F_2 = 1$$

$$F_{n+2} = F_{n+1} + F_n$$

Élément de Cours : Sauvegarder les figures

Sauver et imprimer des figures

Pour imprimer des figures, c'est simple, il suffit de taper la commande `print` (sous unix).

Pour sauvegarder le graphique, la même commande `print` est utilisée. La commande générale est :

```
print('nom.extension', '-dextension')
```

De nombreux formats sont disponibles : gif, jpg, png, eps, svg.... La figure est sauvée dans le répertoire courant.

```
print('graphe.jpg', '-djpeg')
```

Exercice 7 : Résoudre et afficher des équations différentielles

GNU Octave permet aussi de résoudre des équations différentielles ordinaires du 1^{er} ordre [ODE en anglais] et par voie de conséquence de visualiser les résultats.

La définition d'un tel système repose sur la définition de n fonctions de $n+1$ variables (forme de Cauchy).

Ces fonctions seront programmées dans une fonction GNU Octave sous la forme canonique suivante :

```
function ypoint = f(t, y)
    ypoint(1) = une expression de y(1), y(2) ... y(n) et t
    ...
    ypoint(n) = une expression de y(1), y(2) ... y(n) et t
    ypoint = ypoint(:);
end
```

Ensuite, il faut appeler un solveur et lui transmettre *a minima* : le nom de la fonction, les bornes d'intégration t_{\min} et t_{\max} et les conditions initiales.

Habituellement, les solveurs `ode45` (Méthode de Runge-Kutta-Merson d'ordre 4,5) ou `ode23` (d'ordre 2,3) sont utilisés mais il en existe d'autres 😊 GNU Octave prend par défaut une erreur relative maximale de 10^{-4} par défaut (mais cela peut être modifié).

Les fonctions `ode` renvoient un vecteur colonne représentant la variable t et une matrice y dont les colonnes sont les solutions.

Nous souhaitons par exemple résoudre l'équation différentielle du second ordre suivante (oscillateur harmonique sans frottements) :

$$\ddot{x} + \frac{k}{m}x = 0 \quad (\text{avec } k, \text{ constante de raideur et } m, \text{ masse du ressort})$$

Afin de résoudre cette équation, il est nécessaire de l'exprimer sous forme vectorielle. On définit donc 2 vecteurs :

- x avec $x(1) = x$ et $x(2) = dx/dt$
- $dxdt$ avec $dxdt(1) = dx/dt$ et $dxdt(2) = d^2x/dt^2$

Ce qui nous donne au final : $dxdt(1) = x(2)$ et $dxdt(2) = -k/m \cdot x(1)$

Nous allons remplir ces deux dernières lignes dans un fichier `oscillateur.m`

```
function dxdt=oscillateur(t,x)

    k=2 ;
    m=10 ;
    dxdt(1)=x(2)
    dxdt(2)=-(k/m)*x(1)

    dxdt=dxdt' % on convertit la ligne en colonnes

end
```

On définit maintenant un fichier `equadiff.m` dans lequel on définit le domaine d'étude t ($t_{\text{initial}}=0$ et $t_{\text{final}}=100$) et on définit les conditions initiales $dxdt(0)=0$ et $x(0)=0,25$ et où on appelle le solveur.

Ex : `[t,x] = ode45(@oscillateur,[0 100], [0, 0.25])`

Nota : `x(:,1)` contient $x(t)$ et `x(:,2)` contient (dx/dt)

Exercice : Résoudre l'équation différentielle ci-dessus et afficher les courbes (x, t) ainsi que le plan de phase $(x(t), dx/dt)$ dans une sous-fenêtre.

Exercice 8 : Utiliser des données issues de capteurs

Nous souhaitons visualiser des données enregistrées par un accéléromètre 3 axes. Téléchargez le fichier '`accelero.csv`' à l'adresse suivante :

<https://github.com/truillet/upssitech/blob/master/GCGEO/1A/TP/accelero.csv>

Le fichier contient les données (*temps, G_x , G_y , G_z , G*)

1. Charger le fichier dans Octave en utilisant la commande `csvread` (`csvread("accelero.csv")`), et définissez les vecteurs X, Y, Z à partir du fichier chargé
2. Afficher les fenêtres suivantes : X, Y en fonction de t (en bleu), X, Z en fonction de t (en vert)
3. Sauver votre résultat dans un fichier image « `png` »
4. Ecrire un script octave (ou une fonction) qui permet d'effectuer ces opérations automatiquement

(Optionnel) : enregistrer vos propres données issues de capteurs et réeffectuer ces opérations.