I – Introduction

Introduction

Les cinq séances de travaux pratiques de ce module vous permettront d'expérimenter numériquement avec les concepts vus en cours et TD de façon à leur apporter un éclairage plus concret ; inversement, la théorie vous permettra de comprendre ce qui se passe en pratique. Assurez-vous donc de tâcher de faire les liens entre les différents points de vue (cours / TD / TP) sur les notions abordées : le tout forme un ensemble cohérent.

En fin de semestre aura lieu une épreuve individuelle permettant d'évaluer les compétences développées lors des 5 séances ; assurez-vous donc de profiter de celles-ci pour comprendre ce qui s'y passe.

Mode opératoire proposé : pendant les séances, réunissez-vous en petits groupes de travail (environ 4 étudiants semble une taille raisonnable) pour avancer conjointement sur le sujet tout en discutant des concepts et difficultés rencontrées. Vous séparer le travail « moi je fais la a) . . . » ne semble **pas** une bonne idée, à moins que vous ne vous satisfassiez de maîtriser environ le quart du contenu.

« Pour apprendre à patiner, mieux vaut passer du temps sur la glace à s'entraîner à patiner que dans les gradins à regarder ceux qui patinent. »

Pour garder une trace de votre travail lors de ses séances, nous vous demandons de tenir un **journal de bord numérique** dans lequels vous consignerez de façon structurée et (re)lisible :

- les comptes-rendus de vos expérimentations et résultats (code, figures, . . .) ;
- vos réponses aux questions, observations, conclusions
- toute autre information jugée digne d'intérêt (références externes, . . .).

Cela a pour but d'encourager de bonnes pratiques de documentation et d'archivage qui facilitent grandement la vie de tout ingénieur ou scientifique au sens large : reproductibilité des résultats, rédaction de rapport, . . . ou simple consultation personnelle subséquente (par exemple lors de l'évaluation de TP).

A – Premiers pas avec MATLAB

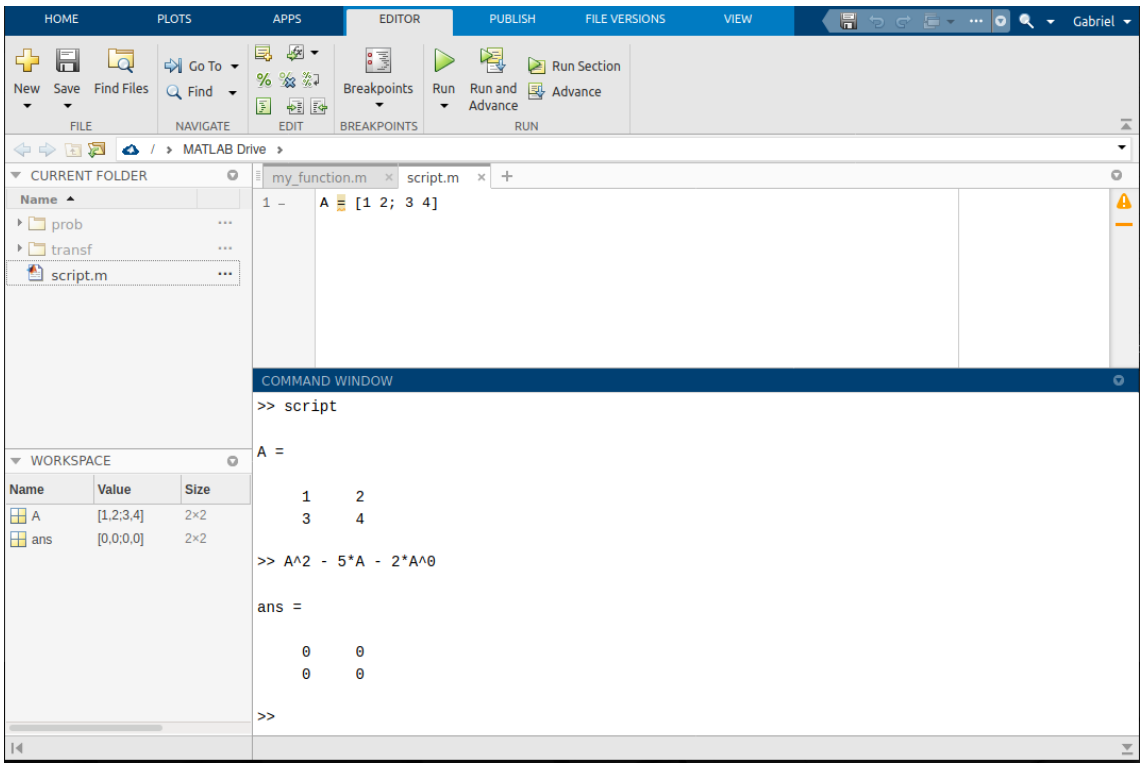
Nous allons utiliser pour les séances de travaux pratiques le logiciel MATLAB, standard de l'industrie en science et ingénierie pour le calcul numérique. Assurez-vous pour commencer d'avoir accès à la dernière version (R2021b) du logiciel, que ce soit localement, sur votre système d'exploitation préféré, ou en ligne.

Les instructions d'accès se trouvent dans le canal général de *Teams*.

L'utilisation de versions antérieures, ou, mieux, de l'alternative libre GNU Octave, est à vos risques et périls car non systématiquement testée (mais ne devrait normalement pas trop poser problème).

Au démarrage, vous devriez obtenir une interface graphique semblable à celle-ci, composée de :

- un explorateur de fichiers,
- un espace de travail affichant les variables courantes ainsi que leur valeur,
- ainsi que le panneau principal dans lequel s'afficheront les différentes fenêtres de travail.



Vous voilà donc devant une calculatrice (certes extrêmement élaborée). Si l'objectif de ces TPs n'est pas de faire de vous des MATLABeurs chevronnés, voici néanmoins quelques points à savoir.

♦♦♦ Attention : bien prendre le temps de lire cette section, et **tester, expérimenter** les commandes *même si vous avez l'impression que ça n'en vaut pas la peine* ; l'expérience montre que la réponse à plusieurs des questions qui se poseront au cours des prochaines heures/semaines s'y trouve !

La documentation officielle

. . . se trouve ici ; l'aide ponctuelle sur une commande s'obtient *via*

```
help nomDeLaCommande
```

Invite de commande

vous pouvez discuter avec MATLAB en entrant des commandes dans le terminal à l'invite `>>` :

```
n = pi + exp(1)   % affecte à n la valeur pi + e
```

Vous remarquez déjà que :

- π et la fonction exponentielle sont déjà connus ;
- le résultat en donné sous forme d'une approximation numérique (et non symbolique) ;
- il n'est pas obligatoire de déclarer une variable avant lui affecter une valeur ;
- le symbole « `%` » sert à mettre en commentaire le reste de la ligne ;
- si vous ne voulez pas afficher le résultat d'une commande, mettez un « `;` » à la fin de la ligne.

Scripts utilisateurs

Une fois que vous avez trouvé les bonnes commandes pour résoudre un problème, vous pouvez les recopier dans un fichier d'extension `.m` (disons par exemple `tp1question1.m`) situé dans votre répertoire de travail

et exécuter l'ensemble de ces commandes d'un seul coup en appelant le fichier :

```
tp1question1  % sans l'extension !
```

Cela permet de résumer votre travail dans un document pouvant facilement être partagé entre collègues.

Gestion optimisée des tableaux

Il faut savoir que MATLAB a été spécialement conçu pour le calcul MATriciel. La syntaxe pour déclarer une matrice M à m lignes et n colonnes remplie de zéros est

```
M = zeros(m,n);
```

Vous avez de même les commandes `ones(m,n)`, `rand(m,n)`, `eye(n)`... Mais surtout, on peut facilement construire des vecteurs de valeurs consécutives :

```
T = [5:.5:8]  % on peut omettre les crochets
```

De façon générale, vous l'aurez compris, `[d:p:f]` produit un vecteur de valeurs démarrant à `d`, ne dépassant pas `f`, avec un pas `p` (égal à `1` si omis, la syntaxe étant alors seulement `[d:f]`).

Ce sont les tableaux qui permettent d'implémenter les boucles *for*. ; elle permet aussi d'accéder à certains éléments d'un tableau : si `T(4)` renvoie le 4e élément de `T`, la commande `T(2:6)` renvoie le sous-tableau comportant les éléments d'indices 2, 3, 4, 5 et 6 de `T`.

♦♦♦ Attention : les matrices sont indicées à partir de 1 (et non 0 comme dans certains langages).

De même, si $M := \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$, alors

```
M=[1 2 3 4; 5 6 7 8; 9 10 11 12]  % le ";" sépare les lignes
M([1 3], [2:4])                   % éléments en ligne 1 et 3, et en colonne 2 à 4\\
```

renvoie $\begin{bmatrix} 2 & 3 & 4 \\ 10 & 11 & 12 \end{bmatrix}$.

Enfin, pour effectuer des opérations « composante à composante » dans un vecteur (par exemple, vouloir calculer les carrés des éléments de `T`), on utilisera `T.*T` ou `T.^2`. Sans le point (« `.*` »), MATLAB tenterait une multiplication matricielle.

Tracé de graphes de fonctions

Les vecteurs servent aussi à tracer des fonctions. Il suffit de définir dans un premier tableau les abscisses et dans le second les ordonnées, par exemple ici pour la fonction $x(t) = t \sin t$ sur $[-5, 10]$:

```
t = -5:.01:10;              % de -5 à 10 par pas de 0,01
x = t.*sin(t);              % vecteur de valeurs (notez le ".*")
figure(1)                   % ouvre ou ré-ouvre la figure numéro 1
plot(t,x)                   % affiche le graphe
hold on                     % les graphes suivants se superposeront
plot(t,cos(t),"r")          % ajoutons un cosinus rouge
```

La commande `plot(t,x)` trace `x` (ordonnées) en fonction de `t` (abscisses).

Pour effacer la figure, faites `clf` ; pour afficher une grille : `grid on`, pour le reste, tapez help.

B – Régression linéaire et quadratique

Commencez par importer dans MATLAB le fichier `data.mat` contenant $n = 2021$ paires d'observations expérimentales (x_i, y_i) (assurez-vous auparavant qu'il soit présent dans le répertoire courant).

```
load data.mat      % ou double-cliquer dessus dans l'explorateur de fichiers
n = size(data, 1)  % 2021 lignes
x = data(:,1);
y = data(:,2);
```

Vous devriez alors voir les vecteurs correspondants s'afficher dans votre espace de travail mémoire.

Depuis le TD1 nous savons comment calculer les coefficients a et b de la droite des moindres carrés associée aux vecteurs $\vec{x}$ et $\vec{y}$:

$$a = \frac{n(\sum x_i y_i) - (\sum x_i)(\sum y_i)}{n(\sum x_i^2) - (\sum x_i)^2} \qquad \text{et} \qquad b = \frac{(\sum x_i^2)(\sum y_i) - (\sum x_i)(\sum x_i y_i)}{n(\sum x_i^2) - (\sum x_i)^2}.$$

```
den = n * sum(x.^2) - sum(x)^2;
a = (n * dot(x,y) - sum(x) * sum(y))/den
b = (sum(x.^2) * sum(y) - sum(x) * dot(x,y))/den
```

Il y a une façon plus simple sous MATLAB d'obtenir le même résultat (sans avoir à connaître ces formules). On cherche après tout des constantes a et b pour lesquelles on aurait approximativement

$$y_i \approx a x_i + b, \qquad i = 1, \ldots, n,$$

soit, sous forme vectorielle (en notant $\vec{1}$ le vecteur-colonne constant ne contenant que des 1) :

$$\vec{y} \approx a\,\vec{x} + b\,\vec{1} = \underbrace{\begin{bmatrix} \vec{x} & \vec{1} \end{bmatrix}}_{A} \begin{bmatrix} a \\ b \end{bmatrix}.$$

Or, lorsqu'on demande à MATLAB de résoudre un système surdéterminé comme celui-ci (2021 équations, 2 inconnues), son comportement par défaut est de renvoyer la meilleure solution *au sens des moindres carrés* (même si le système n'est pas *stricto sensu* compatible). C'est exactement ce que l'on souhaite !

```
A = [x x.^0];  % matrice n x 2 contenant x en première colonne et des 1 en deuxième
coeff = A\y;   % on isole coeff dans l'équation y = A*coeff
a = coeff(1)
b = coeff(2)   % on obtient bien les mêmes valeurs
```

Voilà qui est très bien, nous disposons maintenant de la *meilleure* droite

$$y = ax + b$$

représentant les 2021 données initiales. Il s'agit de la base de l'intelligence artificielle : « comprendre » les données signifie s'en donner un modèle simple. Mieux vaut effectivement stocker en mémoire l'équation d'une droite (2 nombres) que de l'ensemble des valeurs observées (4042 nombres).

Il est grand temps ceci dit de faire ce que l'on aurait dû faire dès le départ : *observer le jeu de données.*

```
clf; plot(x,y,".","markersize",4)   % les données
hold on
I = min(x):max(x);
plot(I,a*I + b,"r","linewidth",4)   % la droite de régression
hold off
```

Ce n'est pas fameux ! Il faut dire que les points expérimentaux ne sont pas spécialement alignés – on semble plutôt être en présence d'une dépendance *quadratique* centrée en $x \approx 45$.

1) Confirmez cette impression effectuant une régression linéaire entre `xx` et `y`, où

```
xx = (x - 45).^2;
```

Observez comment la courbe de régression $y = a(x-45)^2 + b$ ainsi obtenue se compare aux données.

En fait, plutôt que de mettre la constante 45 « en dur » dans les équations, il vaut mieux demander à MATLAB de nous le trouver en projetant les données sur le sous-espace engendré par les constantes, les variables et leurs carrés.

```
A = [x.^2 x x.^0];
coeff = A\y;
d = coeff(1)
e = coeff(2)
f = coeff(3)
```

On obtient ainsi la *meilleure* parabole $y = dx^2 + ex + f$, au sens des moindres carrés, représentant notre jeu de données.

2) Comment se compare-t-elle à la première approximation quadratique obtenue ci-dessus ? Observer notamment l'abscisse du sommet de la parabole $(-\frac{e}{2d})$.

C – Fonctions continues

On sait que lorsque l'on travaille avec des fonctions (réelles) continues sur un intervalle $[a, b]$, la notion naturelle de produit scalaire est

$$\langle x \,|\, y \rangle = \int_a^b x(t)\, y(t)\, \mathrm{d}t.$$

Or cette intégrale peut être approximée par une somme de Riemann associée à une partition de l'intervalle $[a, b]$ en n segments égaux :

$$a = t_0 < t_1 < \cdots < t_n = b \qquad \text{avec} \qquad \Delta t = t_{i+1} - t_i = \frac{b-a}{n}$$

c'est-à-dire

$$\langle x \,|\, y \rangle \approx \sum_i x(t_i)\, y(t_i)\, \Delta t = \Delta t\, (\vec{x} \cdot \vec{y})$$

où $\vec{x} = (x(t_0), \ldots, x(t_n))$ et $\vec{y} = (y(t_0), \ldots, y(t_n))$ sont les versions échantillonnées des fonctions x et y.

Comme en MATLAB tout est vecteur et que $\vec{x}$, $\vec{y}$ sont exactement ce que nous manipulons en lieu et place des fonctions symboliques, cela signifie que le produit scalaire au sens des fonctions se calcule aisément (à une constante multiplicative près) par le produit scalaire usuel `dot` sur les vecteurs.

1) Utilisez MATLAB pour calculer numériquement l'angle entre les fonctions $x(t) = 1$ et $y(t) = t$ sur l'intervalle $[0, 1]$. Comparez votre réponse avec celle obtenue dans le cours.

2) En réutilisant les méthodes de la section B, déterminez la meilleure approximation quadratique de la fonction e^t sur l'intervalle $[-1, 1]$:

$$e^t \approx a + bt + ct^2 \qquad \text{pour } t \in [-1, 1].$$

Superposez e^t et son approximation sur un même graphe et discutez de la qualité de l'approximation.