Université Pierre et Marie Curie - Paris VI

Outils Mathématiques et utilisation de Matlab

Cours 2013-2014

Licence Professionnelle (L3)

Instrumentation Optique et Visualisation

Quentin Glorieux
quentin.glorieux@lkb.upmc.fr

Table des matières

Avant-Propos

Il s'agit des notes de cours destinées aux étudiants de Licence Professionnelle LIOVIS de l'UPMC. Ce cours suppose une familiarité avec les notions de bases d'algèbre, d'analyse et de statistiques. Le format de ce cours est composé de 4 séances de 4h de cours associées à 4 séances de 4h de travaux pratiques. Chaque couple de séances (cours-TP) porte sur un thème différent associé à un chapitre de ces notes. L'utilisation de logiciel de calcul est devenu depuis les années 70 absolument indispensable dans le domaine scientifique, aussi bien pour le technicien que l'ingénieur ou le chercheur. `Matlab` a été développé par la société *Mathworks*. Il s'agit d'un des langages de programmation scientifique les plus populaires.

Des alternatives libres existent comme par exemple Scilab, R ou Octave.

Dans ce cours nous allons parcourir les bases de Matlab pour **analyser des données** efficacement, et **simuler numériquement** des systèmes physiques. Il faut bien comprendre la différence entre ces deux objectifs. D'une part une série de données est produite par une expérience et notre objectif est d'analyser ces données pour en faire ressortir les éléments les plus pertinents. Lorsque l'on parle de donnée, il peut s'agir par exemple d'une mesure d'un paramètre physique, du cours de la bourse, des réponses à un sondage ou bien de tout autres valeurs pouvant être tabulées.
Lorsque l'on parle de simulations numériques, on s'intéresse à la modélisation d'un système ce qui passe le plus souvent par la résolution d'équations différentielles.
Pour ces deux objectifs, nous allons utiliser des notions de mathématiques qui seront parfois nouvelles pour le lecteur. Ce cours n'a pas prétention à être un cours formel et rigoureux du point de vue mathématiques et je présenterai donc uniquement les points nécessaires à la compréhension des concepts sans entrer dans les détails. Lorsque cela sera nécessaire, j'indiquerai les références bibliographiques permettant de traiter ces notions mathématiques plus en profondeur.

La particularité de Matlab est de manipuler uniquement des variables numériques (pas de calcul formel). Par défaut ces variables sont codées sur 64 bits (double précision) et peuvent contenir jusqu'à 16 chiffres significatifs. Les nombres les plus grands manipulés par Matlab sont 10^{306}, au delà de cette limite ils seront considérés comme infini et notés `Inf`. Réciproquement, les nombres les plus petits sont 10^{-306}, au delà de cette limite ils seront considérés comme zéro et notés `0`. Lorsqu'une valeur est manquante ou si le résultat d'un calcul n'existe pas ($\frac{0}{0}$ par exemple), la variable prendra la valeur `Nan`.

Nous verrons que les problèmes d'arrondi peuvent s'avérer important lors que nous étudierons la stabilité des méthodes de résolution d'équations différentielles.

Ce cours est divisé en 2 parties. Dans le **chapitre 1**, je vais introduire l'interface de Matlab et nous écrirons le premier programme afin d'afficher un message simple. Nous étudierons ensuite la structure des données ainsi que les opérations de base $(+, -, *, /)$. Une part importante de ce chapitre sera consacré à la présentation de données sous forme de graphiques en 2D et 3D, ce qui nous simplifiera la tache pour de nombreux exercices par la suite. Je ferai également quelques rappels de statistiques élémentaires. Enfin, ce chapitre se termine par la présentation du produit matriciel et son utilisation.

Le **chapitre 2** a pour but d'introduire les notions essentielles au traitement de données. L'analyse et le traitement de données est un sujet très vaste qui nécessiterais un cours à part entière. Je présenterai ici principalement deux sujets : le filtrage en utilisant les principes de la transformée de Fourier ainsi que la recherche de corrélations dans un ensemble de données.

Chapitre 1

Introduction à Matlab

1.1 La philosophie de Matlab

L'objectif de ce premier chapitre est d'introduire les outils nécessaires à l'utilisation de `Matlab`. En programmation on parle **d'objet** pour décrire de manière générale les concepts que l'on manipulent, de la même manière que l'on parle de variables en mathématiques. L'objet le plus commun dans Matlab est la **matrice**. C'est donc un tableau d'éléments d'un type donné, par exemple un tableau d'entiers, de réels, de texte, de variable logiques...

Le format de la matrice est libre, il se définit par le nombre de lignes et le nombre de colonnes. Une **matrice colonne** (n lignes X 1 colonne) est appelée **vecteur**. Dans ce chapitre nous allons donc apprendre à définir, à afficher et à réaliser des opérations sur ces matrices.

Nous reviendrons en detail sur les formats de données plus loin dans ce cours. En effet c'est un point très important car la précision d'un ordinateur n'est pas infinie et donc tout les nombres utilisés seront des arondis.

Le deuxième point que nous allons aborder est une révision de statistiques de base et leur utilisation dans Matlab. Un vecteur définit précédemment peut contenir des variables qui décrivent une expérience (par exemple les notes d'une classe lors d'un cours de Mathématiques), et il est donc intéressant de connaitre les propriétés statistiques de celui-ci.

Nous aborderons pour finir la représentation graphique sous Matlab. En effet, pour présenter des données, on est très souvent amener à réaliser des graphes. Nous verrons quelques une des possibilités de Matlab dans ce domaine, ce qui nous sera très utile par la suite.

1.2 L'interface de Matlab

Avant de commencer le cours proprement dit, nous allons nous familiariser avec l'interface de Matlab. Selon la version utilisée, l'interface peut changer légèrement mais les points centraux resteront identiques.

1.2.1 Command Window

C'est le terminal dans lequel on doit taper les commandes et sur lequel on verra l'affichage des résultats. Une ligne commence toujours par `>>`. Essayez la commande suivante :

```
>> 1+1
ans =
     2
```

et comparez à la commande :

```
>> 1+1;
```

Deux points sont à noter ici. Le premier est que Matlab définit une variable `ans`, lorsque l'on effectue un calcul. Cette variable (qui veut dire *answer*/réponse), s'affiche sur le terminal lorsque l'on omet le signe `;` en fin de ligne. Ici `ans` est une matrice de taille 1x1 (une ligne par une colonne). Une commande utile lorsque l'on a un doute sur le type de variable que l'on vient de créer est `whos` :

La commande `whos()` peut également être utilisée seule (sans argument). Elle retourne alors la liste des variables.

```
>> whos('ans')
  Name      Size            Bytes  Class     Attributes

  ans       1x1                 8  double
```

Cette commande retourne le nom de la variable (ans), sa taille (1x1), le nombre d'octets utilisés (8), et le type de données (ici réel en double précision).

Une seconde commande utile est `clc`. Cette commande permet de vider l'écran de Command Window pour repartir sur une fenêtre propre.

De plus, Matlab supporte l'auto-complétion, c'est à dire que si vous connaissez le début d'une commande, vous pouvez n'entrer que les premières lettres et utiliser la touche `Tab` pour chercher parmi les commandes commençant ainsi.

1.2.2 Workspace

Dans cette fenêtre, on obtient la liste des variables connues par `Matlab` . Il est possible de double-cliquer sur une variable pour l'afficher. Un clic-droit sur les variables offre de nombreuses options telles que : Copiez, Collez, Supprimez etc.

1.2.3 Command History

Lorsque l'on effectue une analyse de données sur les résultats d'une expérience il est **essentiel** de conserver une trace de toutes les opérations qui ont été réalisées. C'est la base de la traçabilite et de la reproductibilité des résultats scientifiques. Il est possible de cliquer sur une commande dans cette fenêtre pour l'exécuter a nouveau. On peut également remonter dans la liste de commandes en se plaçant dans la Command Window et en pressant les flèches de direction. Matlab est également capable de remonter dans la liste de commandes en ne prenant en compte que les commandes qui commencent par certains caractères. Si l'on recherche une commande que l'on a entrées précédemment, on peut taper le début de cette commande puis la flèche du haut et Matlab cherchera uniquement parmi les commandes ayant le même début.

Par ailleurs, il est parfois important d'enregistrer dans un fichier indépendant la liste des commandes utilisées. Pour ce faire on utilise :

```
>> diary filename
```

Cette commande va créer un fichier `filename` qui enregistrera toutes les commandes entrées jusqu'à ce que l'on utilise `diary off`.

1.2.4 Current Folder

Je viens de le décrire la commande `diary` permet de créer un fichier. On peut se poser la question légitime d'où Matlab va stocker ces données. Il est très important de bien maitriser ce point si l'on souhaite ne pas se perdre dans l'ensemble des fichiers auxquels nous allons faire appel via Matlab. Matlab définit ce qui s'appelle le `PATH` (chemin). C'est les dossiers dans lesquels Matlab va chercher lorsque l'on appelle un commande. Le `PATH` est divisé en deux sous-parties : d'une part le `MATLABPATH` et d'autre par le `USERPATH`.
Le premier (`MATLABPATH`) correspond aux différents dossiers auxquels Matlab va faire référence pour utiliser des fonctions prédéfinies par Matlab. Il s'agit si l'on veut des chemins par défaut dans lesquels Matlab cherche les fonctions et il peut contenir un grand nombre de dossiers. L'exemple de code suivant permet d'afficher le `MATLABPATH` :

```
>> path
```

Le second (`USERPATH`) est un unique dossier qui est propre à l'utilisateur lors d'une session. Il est fortement conseillé de définir le `USERPATH`, immédiatement lorsque l'on commence une session Matlab. Le fichier `diary` par exemple est sauvé dans le `USERPATH`. Deux méthodes peuvent être utilisée pour modifier le `USERPATH`. En ligne de commande :

Le `USERPATH` est un sous ensemble du `MATLABPATH` qui doit être personalisé à chaque session. C'est la première ligne lorsque l'on entre la commande `path`.

```
>> newpath = 'C:\Research_Project';
userpath(newpath)
```

ou bien en sélectionnant `File> Set Path` dans les menus déroulants.

La fenêtre Current Folder affiche donc le contenu du `USERPATH`. On peut naviguer entre les dossiers en utilisant la même nomenclature que sous Unix. On peut tester notamment les commandes suivantes : `cd, ls ,cd .., mkdir('newfolder')`. Respectivement ces commandes permettent de changer de dossier, de lister les fichiers du dossier, de remonter d'un niveau dans l'arborescence, et de créer un dossier `newfolder`. On peut tester le code suivant pour créer un nouveau dossier puis l'ajouter au path :

```
>> mkdir('c:/\texttt{Matlab }/myfiles')
addpath('c:/\texttt{Matlab }/myfiles')
```

Une autre application très importante de la fenêtre Current Folder est d'afficher la listes des fichiers de données qui peuvent être chargés par l'utilisateur. En effet, très souvent les données à analyser seront générées par un autre logiciel puis importées dans Matlab via la commande `importdata(filename)`.

L'importation de données étant un point fondamental nous le traiterons en détail par la suite.

1.2.5 Editor

La plupart de votre travail sous Matlab va consister à créer ou modifier des fichiers **.m** qui est le suffixe standard pour les procédures Matlab. Lorsque l'on réalise une tache sous Matlab, il est très souvent possible de le faire en utilisant uniquement la Command Window. Cependant lorsque cette tache devient plus complexe (plusieurs dizaines de ligne de code) ou que l'on souhaite pouvoir la transmettre à quelqu'un d'autre simplement, on utilise la fenêtre Editor. On crée un fichier .m qui peut être au choix un **script** ou une **fonction** (function en anglais). Un script est une suite de commande que l'on aurait tout aussi bien pu taper dans la Command Window. Une fonction permet d'étendre les possibilités au delà des fonctions préprogrammées par les développeurs de `Matlab` .

Par exemple, on pourra réaliser une fonction `racineplus2(input)` qui à un paramètre d'entrée `input` va répondre $\sqrt{\text{input} + 2}$: Pour créer un fichier .m on peut soit utiliser les menus contextuels, soit entrer la commande : `edit FileName.m`. On entre ensuite le code suivant :

La syntaxe de la fonction n'est pas le sujet ici. Nous reviendrons en détails sur cela plus tard ainsi que sur la commande `sqrt()`

```
function [ output ] = racineplus2( input )
output=sqrt(input+2);
end
```

Lorsque l'on sauve cette fonction sous la forme d'un fichier .m, il est important de nommer le fichier `racineplus2.m` et de sauver ce fichier dans le `USERPATH`. On peut ensuite appeler cette fonction simplement dans la Command Window par :

```
>> racineplus2(7)
ans =
     3
```

Il est évident que cette fonction ne fait pas partie des fonctions de base de Matlab : pourquoi choisir +2 et pas +3. Cependant si l'on souhaite l'utiliser souvent il est utile de définir une **fonction utilisateur** qui sera rapide à utiliser. Une fois que l'on a créé un fichier .m, il est possible d'accéder à l'éditeur en double-cliquant sur le nom du fichier dans la fenêtre Current Folder.

1.2.6 Help

Le menu d'aide de Matlab est une des bases de son succès. En effet, l'aide est essentielle lorsque l'on programme avec un langage de haut-niveau comme Matlab, où le nombre de fonctions est très important et la syntaxe est parfois complexe. Pour accéder à l'aide on peut au choix sélectionner une fonction et presser F1, taper `help FunctioNname` ou utiliser les menus déroulants. L'aide doit être vu comme complémentaire de ce cours. Ici j'explique les outils mathématiques que nous utiliserons, dans l'aide de Matlab vous trouverez la syntaxe des différentes fonctions. Mais j'insiste : il est essentiel que vous vous familiarisiez avec les outils de l'aide de Matlab pour réussir dans ce cours.

1.3 "Hello World"

La tradition d'utiliser "Hello World" comme message de test a été initiée par Brian Kernighan et Dennis Ritchie dans le livre *The C Programming Language* publié en 1978.

Tout cours de programmation qui se respecte commence par l'exemple "Hello World". Il s'agit d'un bref programme pour mettre en place les différents éléments nécessaires. Nous allons réaliser deux exemples de ce programme pour mettre en évidence la différence entre un script et une fonction.

1.3.1 Script

Le script est le fichier .m le plus simple. Il s'agit simplement d'une liste de commandes mises bout à bout et sauvegardée dans un fichier. Pour commencer on fixe le `USERPATH`. On crée ensuite un fichier .m dans ce dossier et on nomme ce fichier `hello.m`. On édite ensuite le fichier .m de la façon suivante :

```
str='Hello world';
str
```

Sauvez ensuite ce script. Puis dans la fenêtre Command Window, on tape la commande : `hello`.
Voila nous avons fait le programme le plus simple possible de Matlab, voyons comment nous pouvons l'améliorer.

1.3.2 Fonction

Une fonction va permettre de rentrer des arguments en entrée et d'obtenir différentes variables en sortie. On va essayer ici de modifier le script `hello.m` pour en faire une fonction qui prend votre prénom en entrée et retourne `Hello Votreprénom` en sortie. On commence de façon similaire à pour un script : on crée un fichier .m que l'on nomme `hello2.m`.

```
function [ str ] = hello2 ( prenom )
str = ['Hello ', prenom ];
end
```

On sauvegarde le fichier .m puis on l'appelle depuis la fenêtre Command Window. Cette fois la fonction a besoin d'un paramètre en entrée, on tape donc : `hello2('Quentin')`. On obtient l'affichage voulu. La syntaxe d'une fonction est relativement simple et doit être connue. On définit ce que va retourner la fonction : `function [output1,output2 ... outputN]`. Puis on nomme la fonction, ici : `= hello2()`. On définit alors les paramètres d'entrée `hello2(input1, imput2, ... inputM)`. On peut passer alors au corps même de la fonction, qui doit contenir une définition de toutes les variables de sortie `output1,output2 ... outputN`. Finalement, une fonction se conclut toujours par `end`.

1.4 Outils de base

On l'a déjà dit, le principe de base de Matlab est de considérer la plupart des objets comme des matrices. Ainsi les opérations usuelles $+, -, *, /$ doivent se comprendre comme des opérations matricielles. On consacrera la section suivante à ces opérations. Nous allons dans un premier temps regarder ce qu'ils se passent pour les matrices 1x1 (c'est à dire un seul élément), puis pour les matrices 1xn ou nx1 (c'est à dire des vecteurs ligne ou colonne).

1.4.1 Types de variables

Il existe cinq grands types de variables sous Matlab : les entiers, les réels, les complexes, les chaînes de caractères et le type logique. Définissons une variable de chaque type :

```
>> a = 1.3; b = 3+i; c = 'bonjour';
>> d1 = true(1==1); d2 = logical(1);
>> e = int8(2);
```

a représente un réel, b un complexe, c une chaine de caractères, d1 et d2 sont deux manières de définir une variable logique (VRAI dans le cas présent) et e est un entier codé sur 8 bits. On peut alors vérifier le type de ces différentes variable en utilisant la fonction `whos` :

On peut définir la partie imaginaire d'un complexe en utilisant au choix `i` ou `j`.

```
>> whos
  Name      Size            Bytes  Class      Attributes

  a         1x1                 8  double
  b         1x1                16  double     complex
  c         1x7                14  char
  d1        1x1                 1  logical
  d2        1x1                 1  logical
  e         1x1                 1  int8
```

Il n'est donc pas nécessaire (impossible en fait) de déclarer le type de variable lorsque l'on crée une variable dans Matlab. Il peut alors s'avérer utile de vérifier quel est le type d'une variable. On utilise les fonctions `ischar,islogical,isreal`.

1.4.2 Précision

Avant de poursuivre, je vais mentionner quelques une des limites qu'il est important de connaitre lorsque l'on fait du calcul numérique. Il s'agit du réel le plus grand et le plus petit qui peuvent être manipulé par Matlab, ainsi que de la précision sur ces derniers. Les constantes `realmax` et `realmin` renvoient respectivement le plus grand (petit) nombre à virgule flottante manipulable. La constante `eps` renvoie la précision numérique relative.

```
>> min=realmin
max=realmax
precision=eps

min = 2.2251e-308
max = 1.7977e+308
precision = 2.2204e-16
```

10^{308} avec une précision de 10^{-16}, vous allez me dire pourquoi s'intéresse t'on à cela ? Pour en comprendre l'importance revenons à la manière dont un ordinateur code des nombres entiers et réels. Un ordinateur travaille uniquement en binaire (base 2).Les nombres entiers sont donc codés en binaire sur un nombre plus ou moins important de bits.Si l'on décide de coder un entier naturel (donc non négatif) sur 8 bits, on aura accès à tout les entiers entre 0 et $2^8 - 1 = 255$. De même si désormais on souhaite travailler avec des entiers relatifs, toujours sur 8 bits, alors

Pour definir ou convertir un nombre en entier on utilise `uint` ou `int` pour les entiers naturel ou relatifs suivi du nombre de bits. Par exemple `uint16` signifie un entier naturel codé sur 16 bits.

on aura accès aux entiers entre $-2^7 = -128$ et $2^7 - 1 = 127$. Matlab supporte les entiers jusqu'à 64 bits.

Passons maintenant aux nombres réels. Il existe deux types principaux de codage pour les nombres réels : simple précision (`single`) ou double précision (`double`). Ces deux codages utilisent le principe de la virgule flottante (`float`). Cela consiste à représenter un nombre par son signe (+1 ou -1) s, les chiffres significatifs (ou mantisse) m et un exposant e. Le nombre s'écrit alors $s*m*10^e$. Lorsqu'il est codé en simple précision (c'est à dire sur 32 bits), le nombre se compose d'un bit de signe, 8 bits d'exposant, et 23 bits pour la mantisse. Si la mantisse est codée sur 23 bits, cela signifie que l'on peut obtenir les nombres entre 0 et $2^{23} - 1 = 8388607$, ainsi un nombre réel codé en simple précision aura environ 7 chiffres significatifs (ce qui est parfois relativement peu). En double précision (c'est à dire sur 64 bits), le nombre se compose d'un bit de signe, 11 bits d'exposant, et 52 bits pour la mantisse. Dans ce cas, on obtient environ 16 chiffres significatifs (c'est le sens de la constante `eps`).

Bien choisir son type de données permet d'optimiser la mémoire utilisée ainsi que la rapidité et la précision des calculs numériques. Il s'agit donc d'un critère important lors de l'évaluation de la qualité d'un code.

Un dernier point est le nombre de chiffres lors de l'affichage à l'écran. Le nombre de chiffres retenus pour l'affichage n'est pas nécessairement le même que celui retenu pour la précision (il ne peut pas être plus grand c'est évident !). Les deux formats les plus utilisés sont `short` (défaut) et `long` qui correspondent à 5 et 15 chiffres respectivement.

1.4.3 Arithmétique et opérations sur les scalaires

Nous allons nous intéresser aux opérations mathématiques de bases avec des matrices 1x1, c'est à dire des nombres. Commençons par les 4 opérations que vous connaissez depuis l'école primaire : $+, -, *, /$.

Avant de commencer, je conseille d'utiliser les trois commandes suivantes : `clc` pour nettoyer l'écran, `clear all` pour supprimer toutes les variables créées auparavant, et `close all` pour fermer toutes les fenêtres inutiles.

```
>> x = 1+1
x = 2
```

On peut également travailler avec des variables définies par l'utilisateur :

```
>> x = 2; y = 1.5 ;
somme = x+y
difference = x-y
produit = x*y
division = x/y

somme = 3.5000
difference = 0.5000
produit =  3
division = 1.3333
```

Matlab est sensible à la casse : la variable ToTo est différente de tOtO

On peut également utiliser les fonctions trigonométriques, puissance, logarithmiques etc

```
>> x = 2; y = pi;
cos(y)
exp(x)
sqrt(x)

ans =  -1
ans = 7.3891
ans = 1.4142
```

Voici une liste (non exhaustives) des fonctions incorporées dans Matlab :

`exp(x)` : exponentielle de x

`log(x)` : logarithme néperien de x

`log10(x)` : logarithme en base 10 de x

`x^n` : x à la puissance n

`sqrt(x)` : racine carrée de x

`abs(x)` : valeur absolue de x

`sign(x)` : 1 si $x > 0$ et 0 si $x \leq 0$

`sin(x)` : sinus de x

`cos(x)` : cosinus de x

`tan(x)` : tangente de x

`asin(x)` : sinus inverse de x (arcsin de x)

`sinh(x)` : sinus hyperbolique de x

`asinh(x)` : sinus hyperbolique inverse de x

Evidemment les fonction hyperboliques et inverses sont aussi valables pour le cosinus et la tangente

On pourra également utiliser les fonctions d'arrondis :

`round(x)` : entier le plus proche de x

`floor(x)` : arrondi par défaut de x

`ceil(x)` : arrondi par excés de x

ainsi que les fonctions d'arithmétiques :

`rem(m,n)` : reste de la division entière de m par n

`lcm(m,n)` : plus petit commun multiple de m et n

`gcd(m,n)` : plus grand commun diviseur de m et n

`factor(n)` : décomposition en facteurs premiers de n

Enfin lorsque l'on travaille avec des complexe on pourra utiliser :

`conj(z)` : conjugué de z

`abs(z)` : module de z

`angle(z)` : argument de z

`real(z)` : partie réelle de z

`imag(z)` : partie imaginaire de z

Exercices

(1.1) Evaluez les quantités suivantes dans Matlab avec 5 chiffres significatifs :

(a) tanh(e) (b) $\log_{10}(2)$ (c) $\frac{1}{1+\frac{1}{1+\frac{1}{2}}}$

(d) PGCD(48972, 36533112) (e) $e^{10^{2.7}}$ (f) $\cos^{-1}(\frac{\pi}{4})$

(g) PPCM (318,732) (h) $\sqrt{3\pi}$ (i) $\ln \frac{1}{\pi}$.

(1.2) Calculez $\frac{8}{100} - 0.5\left[\frac{4}{10}\right]^2$. Commentez le résultat.

(1.3) Calculez e^{14} et 382801π jusqu'au 15ème chiffres significatifs. Quel est le plus grand ?

(1.4) Quelle est la meilleure approximation de $\sqrt{7}$: $\frac{2709}{2^{10}}$, $\frac{10583}{4000}$ ou $\frac{2024}{765}$?

(1.5) Comparez la division à droite / et la division à gauche \. Commentez.

(1.6) Trouvez la partie réelle et imaginaire des nombres complexes suivants :

(a) $e^{i(3\pi+4)}$ (b) $\frac{1}{1+i}$ (c) $\ln(-1)$.

(1.7) Calculez la norme et l'argument des nombres complexes suivants :

(a) $e^{i(3\pi+4)}$ (b) $\frac{1}{1+i}$ (c) $\ln(-1)$.

(d) $3+7i$ (e) i^3+1 (f) $e^{i\frac{\pi}{5}}$.

1.5 Vecteurs

Pour grouper des éléments de types differents par exemple des caractères et des réels, on utilisera les structures.

Passons maintenant à l'utilisation des vecteurs. Un vecteur sous Matlab est une collection de d'éléments du même type. Un vecteur pourra représenter des valeurs expérimentales ou bien les valeurs discrétisées d'une fonction continue.

1.5.1 Définir un vecteur

La méthode la plus simples pour définir un vecteur est de donner sa description explicite à l'aide de la commande `[ ]`, par exemple :

```
vec = [1 2 4 7 9 2.3]

vec =
    1.0000    2.0000    4.0000    7.0000    9.0000    2.3000
```

On peut également définir un vecteur colonne en utilisant le `;`

```
col = [1 ; 2 ; 4 ; 7]

col =
     1
     2
     4
     7
```

On peut concaténer deux vecteurs :

```
vec1 = [1 3 5];
vec2 = [9 10 11];
vec = [vec1 vec2]

vec =
     1     3     5     9    10    11
```

Et on peut également prendre la transposée pour passer d'une ligne à une colonne ou réciproquement :

```
vec1 = [1 3 5];
vec = vec1'

vec =
     1
     3
     5
```

Il n'est pas nécessaires de définir la taille d'un vecteur (c'est automatique), par contre la commande `length()` permet de retourner cette quantité.

```
length(vec)

ans = 3
```

Lorsque l'on veut afficher le graph d'une fonction f telle que $y = f(x)$ sous Matlab, il faut définir deux vecteurs : un pour le vecteur x, l'autre pour le vecteur $y = f(x)$. Matlab ne fait pas du calcul formel, il faut donc discretiser l'axe des x. On utilise pour ce faire la fonction `linspace(a,b,n)` Cette commande génère un vecteur ligne de n éléments espacés linéairement entre a et b. Par défaut $n = 100$.

Une autre méthode pour générer des vecteurs espacés linéairement consiste à utiliser `[a,s,b]`. On crée alors un vecteur entre a et b avec un espacement s :

```
vec=[1:2:10]

vec =     1     3     5     7     9
```

Il existe enfin des vecteurs spéciaux prédéfinis dans Matlab :

`ones(1,n)` : vecteur ligne de longueur n dont tous les éléments valent 1

`zeros(1,n)` : vecteur ligne de longueur n dont tous les éléments valent 0

`rand(1,n)` : vecteur ligne de longueur n dont les éléments sont générés de manière aléatoire entre 0 et 1.

`ones(m,1)` sera donc vecteur colonne de longueur m dont tous les éléments valent 1 (de même pour `zeros` et `rand`).

1.5.2 Manipuler un vecteur

Il est important également de se familiariser à la manipulation de vecteurs, c'est-à-dire être capable d'extraire des sous-ensembles à l'aide des indices. Le k^{eme} élément d'un vecteur `vec` peut être affiché grâce à la commande `vec(k)`. k doit être un entier sinon Matlab retournera une erreur :

```
>> vec = linspace(1,10,10)
vec = 1     2     3     4     5     6     7     8     9    10

>> vec(4)
ans = 4

>> vec(4.2)
Subscript indices must be real positive integers or logical.
```

On peut également utiliser des vecteurs d'indices pour extraire un sous-vecteur :

```
>> vec = linspace(1,89,9)
vec =     1    12    23    34    45    56    67    78    89

>> vec(3:6)
ans =  23    34    45    56

>> vec(4:2:8)
ans = 34    56    78

>> subvec=[1 3 5]; vec(subvec)
ans = 1    23    45
```

1.5.3 Opérations vectorielles

Les opérations algébriques usuelles $+, -, *, /$ doivent être prises avec précautions pour les vecteurs. La somme et la différence sont des opérations termes à termes, et nécessitent donc des vecteurs de même dimension. Le produit $*$ est le produit matriciel. Nous y reviendrons dans la section sur les matrices. Pour utiliser la mutliplication ou la division termes à termes on doit remplacer $*$ par $.*$ et $/$ par $./$

Pour les exposants on utilise `vec.^2`

```
>> vec = [1 3 5 6]; vec2 = [10 20 30 40];

>> vec+vec2
ans = 11    23    35    46

>> vec-vec2
ans =  -9   -17   -25   -34

>> vec.*vec2
ans = 10    60   150   240

>> vec2.*vec
ans = 10.0000    6.6667    6.0000    6.6667
```

De la même manière que pour les scalaires, on peut appliquer toutes les fonctions définies précédemment pour les vecteurs. Par exemple :

```
>> vec = linspace(1,10,10);
out=sqrt(vec)

out =
  Columns 1 through 5
    1.0000    1.4142    1.7321    2.0000    2.2361
  Columns 6 through 10
    2.4495    2.6458    2.8284    3.0000    3.1623

>> out2=vec.^2
out2 =
  Columns 1 through 5
 1     4     9    16    25
  Columns 6 through 10
 36    49    64    81   100

>> out3=cos(vec)
out3 =
  Columns 1 through 5
 0.5403   -0.4161   -0.9900   -0.6536    0.2837
  Columns 6 through 10
 0.9602    0.7539   -0.1455   -0.9111   -0.8391
```

Il existe aussi des commandes qui sont propres aux vecteurs.

Ces commandes s'appliquent aussi aux matrices. Dans ce cas la commande porte sur chaque vecteur colonne de la matrice.

`sum(x)`	:	somme des éléments du vecteur x
`prod(x)`	:	produit des éléments du vecteur x
`max(x)`	:	plus grand élément du vecteur x
`min(x)`	:	plus petit élément du vecteur x
`mean(x)`	:	moyenne des éléments du vecteur x
`sort(x)`	:	ordonne les éléments du vecteur x par ordre croissant
`fliplr(x)`	:	renverse l'ordre des éléments du vecteur x

Exercice

(1.8) Manipulation de vecteurs

Donnez le code Matlab qui permet de :

a. Créez un vecteur colonne `vec` de 5 éléments linéairement espacés entre 2 et 3.
b. Ajoutez deux lignes à la fin de ce vecteur avec la valeur 0.
c. Ajoutez 1 au deuxième et sixième éléments de ce vecteur.
d. Créez un second vecteur `vec2` colonne de même dimension que `vec` contenant les entiers pairs supérieurs ou égaux à 6.
e. Définir un vecteur `sumvec` comme la somme des deux vecteurs `vec` et `vec2`.
f. Définir un vecteur `prodvec` comme le produit termes à termes des deux vecteurs `vec` et `vec2`.

g. Quel est la somme des éléments de `prodvec` ?
h. Quel est la moyenne des éléments de `sumvec` ?
i. Quel est le plus grand élément du vecteur $v3 = \frac{\texttt{vec}^2+\sqrt{\texttt{vec2}+1}}{\texttt{vec}.(\texttt{vec2}+1)}$?

1.6 Matrices

Nous arrivons maintenant aux matrices ! C'est le coeur de Matlab. Donc on se concentre !!!

1.6.1 Définir une matrice

Une matrice va se définir de façon similaire à un vecteur avec la commande `[]`. On définit la matrice A :

$$A = \begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$$

```
>> A = [1 2 ; 4 3]

A =
     1     2
     4     3
```

Cela est équivalent à

```
>> A = [1 2
        4 3]
A =
     1     2
     4     3
```

Une matrice est composée de m lignes et n colonnes. Si on souhaite connaitre la valeur de m ou n, on utilise la commande `size(A)`

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

>> [m n] = size(A)
m = 2
n = 3

>> size(A,1)
ans =  2
```

On peut construire très simplement une matrice "par blocs". Si A, B, C, D désignent 4 matrices (aux dimensions compatibles), on définit la matrice blocs :

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

par l'instruction `M = [A B ; C D]`.

Comme pour les vecteurs il existe des matrices prédéfinies :

`eye(n)`	:	la matrice identité (carrée de taille `n`)
`ones(m,n)`	:	la matrice à `m` lignes et `n` colonnes dont tous les éléments valent 1
`zeros(m,n)`	:	la matrice à `m` lignes et `n` colonnes dont tous les éléments valent 0
`rand(m,n)`	:	une matrice à `m` lignes et `n` colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1.
`magic(n)`	:	une matrice magique de dimension `n`.

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1

>> ones(2,4)
ans =
     1     1     1     1
     1     1     1     1

>> zeros(2)
ans =
     0     0
     0     0

>> rand(2,3)
ans =
    0.8147    0.1270    0.6324
    0.9058    0.9134    0.0975
```

1.6.2 Manipuler une matrice

Pour extraire un élément de la matrice on indique la ligne et la colonne de celui-ci :

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

>> A(2,1)
 ans = 4
```

Lorsque l'on souhaite extraire une colonne ou une ligne entière on utilise le symbole (:) comme on va le voir dans l'exemple suivant :

```
>> A(2,:)
 ans = 4     3     6

>> A(:,1)
ans =
     1
     4
```

Toutes les combinaisons sont alors possibles. On peut extraire 2 colonnes par exemple en faisant :

```
>>  A(:,[1 2])
ans =
     1     2
     4     3
```

On verra dans les exercices à la fin de cette section que la manipulation de matrices est vraiment très puissante chez Matlab.

Comme pour les vecteur il est possible d'obtenir la transposée d'une matrice avec la commande '.

```
>>  A'
ans =
     1     4
     2     3
     5     6
```

Il existe des commandes matlab permettant de manipuler globalement des matrices. La commande `diag` permet d'extraire la diagonale d'une matrice : si A est une matrice, diag(A) est le vecteur composé des éléments diagonaux de A. La même commande permet aussi de créer une matrice de diagonale fixée : si v est un vecteur de dimension n, `A=diag(v)` est la matrice diagonale dont la diagonale est v.

```
>> A=eye(3)
diag(A)'

A =
     1     0     0
     0     1     0
     0     0     1
ans =
     1     1     1

>> v=[1:3];
>> diag(v)
ans =
     1     0     0
     0     2     0
     0     0     3
```

Exercice

(1.9) Manipulation de matrices

a. Construire la matrice $M = \begin{pmatrix} 12 & \pi \\ 3i+4 & 1 \end{pmatrix}$.
b.Construire la matrice T tridiagonale à l'aide de la commande diag() utilisée 3 fois :

$$T = \begin{pmatrix} 1 & 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

c. Extraire de T les deux premières colonnes.
d. Extraire de T les éléments des colonnes et des lignes 2 à 4.
e. Créer une matrice $T2$ où la ligne 1 est échangée avec la ligne 3 puis la colonne 2 est remplacée par les valeurs de la colonne 4.

1.6.3 Opérations matricielles : addition et soustraction

Nous entrons maintenant dans le premier point où nous allons faire des maths et pas uniquement de la programmation (enfin...) ! Les opérations matricielles sont à la base du calcul sous Matlab. Les concepts mathématiques qui les sous tendent sont ceux de l'algèbre linéaire. L'objectif ici n'est pas de faire un cours complet d'algèbre linéaire, mais de revoir les opérations matricielles et en premier lieu : le produit de matrices et l'inversion. La notion importante de diagonalisation sera traitée en détails dans le chapitre suivant.

Commençons par l'addition et la soustraction. Ces opérations sont possibles uniquement sur des matrices de taille identique. Ce sont des opérations termes à termes, similaires aux opérations scalaires. Par exemple :

$$\begin{pmatrix} 1 & 5 \\ -3 & 2 \end{pmatrix} + \begin{pmatrix} 4 & 2 \\ 0 & -4 \end{pmatrix} = \begin{pmatrix} 5 & 7 \\ -3 & -2 \end{pmatrix}.$$

Sous Matlab, la syntaxe est tout aussi simple, elle reprend les mêmes notations que pour les scalaires et les vecteurs :

```
>> A = [1 5 ; -3 2]; B = [4 2 ; 0 -4];
M = A+B

M =
     5     7
    -3    -2
```

Je ne détaille pas la soustraction qui est exactement identique.

1.6.4 Opérations matricielles : produit

Attention : ce n'est pas la multiplication à laquelle les gens font référence lorsque l'on demande de multiplier deux matrices. La multiplication termes à termes peut parfois être utile mais dans ce cas il sera précisé que ce type de multiplication doit être utilisé !

Par contre la multiplication mérite une attention particulière. Il existe deux types de multiplication : la multiplication dite matricielle et la multiplication termes à termes. Commençons par cette dernière : la multiplication termes à termes est l'analogue de l'addition et de la soustraction vues précédemment. Sous Matlab elle se note de façon spécifique pour la distinguer de la *vraie* multiplication matricielle : `A.*B`

```
>> A = [1 5 ; -3 2]; B = [4 2 ; 0 -4];
M = A.*B

M =
     4    10
     0    -8
```

De même si l'on souhaite obtenir le carré d'une matrice (au sens du produit termes à termes de cette matrice par elle même) on ecrit : `A.^2`

```
>> A = [1 5 ; -3 2 ; 1 3];  M=A.^2

M =
     1    25
     9     4
     1     9
```

Passons, maintenant au véritable produit matriciel. Le produit (non-commutatif) entre la matrice A de taille m x n et la matrice B de taille n x p est une matrice $M = AB$ de taille m x p. Pour que ce produit soit défini, il est nécessaire que le nombres de colonnes de A soit égal au nombre de ligne de B. Si l'on notre les éléments de A : a_{ij}, et ceux de B : b_{ij}, alors les éléments de la matrices M sont donnés par la formule suivante :

$$m_{ij} = \sum_{0<k\leq n} a_{ik}b_{kj}$$

Cette formule traduit un principe relativement simple décrit sur le schéma suivant :

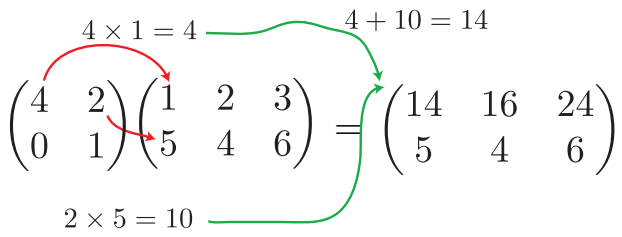


FIGURE 1.1: Représentaion schématique de la multiplication matricielle.

Sous Matlab, on calcule le produit matriciel en utilisant simplement le signe `A*B` :

```
>> A = [4 2 ; 0 1]; B = [1 2 3; 5 4 6]; M=A*B

M =
    14    16    24
     5     4     6
```

Comme je l'ai dit précédemment, il faut faire attention à la taille des matrices pour que le produit soit défini. Par exemple si l'on essaie de faire calculer le produit BA on obtient :

```
>> A = [4 2 ; 0 1]; B = [1 2 3; 5 4 6]; B*A
Error using  *
Inner matrix dimensions must agree.
```

1.6.5 Opérations matricielles : inverse et division

Maintenant que l'on dispose du produit matriciel, on peut définir l'inversion. En effet, on note A^{-1}, l'inverse de A (quand elle existe) et on définit A^{-1} par :

$$A^{-1}A = AA^{-1} = I,$$

où I est la matrice identité. On verra par la suite que cette notion d'inverse est très importante pour résoudre un système d'équations par exemple. En attendant, voila comment Matlab calcule l'inverse d'une matrice :

Seule les matrice carée sont inversibles. La matrice I est de la taille de la matrice A.

```
>> A = [4 2 ; 0 1]; M = inv(A)

M =
         0.2500   -0.5000
         0         1.0000
```

La division se définit à partir de l'inverse :

$$A/B = AB^{-1}.$$

Elle nécessite donc que B soit inversible et que les dimensions de A et B soient compatibles. Elle s'implémente sous Matlab de façon simple :

```
>> A = [4 2 ; 0 1]; B = [1 2 ; 5 4]; M = A/B

M =
   -1.0000    1.0000
    0.8333   -0.1667
```

1.6.6 Résoudre un système linéaire

Nous allons maintenant utiliser le formalisme matriciel pour résoudre un système d'équations. Le système que nous cherchons à résoudre est le suivant :

$$\begin{cases} 3x + 5y + z & = 1 \\ 7x - 2y + 4z & = -3 \\ -6x + 3y + 2z & = 3 \end{cases}$$

On définit alors la matrice M à partir des coefficients de ce système :

$$M = \begin{pmatrix} 3 & 5 & 1 \\ 7 & -2 & 4 \\ -6 & 3 & 2 \end{pmatrix}.$$

Si on note $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, on peut voir que les trois lignes du produit MX correspondent aux trois lignes du système. On peut alors définir la matrice $b = \begin{pmatrix} 1 \\ -3 \\ 3 \end{pmatrix}$, tel que $MX = b$.

M n'est pas toujours inversible, dans ce cas le système n'a pas de solution ou a une infinité de solutions.

Pour obtenir les solutions du système linéaire, il ne reste plus qu'à inverser la matrice M. On a donc :

$$X = M^{-1}b.$$

Il existe deux méthodes pour implémenter ce calcul sous Matlab. La première méthode reprend ce que nous avons appris précédemment. On doit définir la matrice A et la matrice b, calculer l'inverse de A et faire le produit avec b :

```
>> A = [3 5 1 ; 7 -2 4 ; -6 3 2]; b = [1 -3 3]';
>> X=inv(A)*b

X =
   -0.3100
    0.3886
   -0.0131
```

On peut alors vérifier que :

```
>> 3*X(1)+5*X(2)+X(3)

ans =    1.0000
```

La seconde méthode est plus élégante. Elle introduit la division à gauche. Lorsque l'on note sous Matlab `A\b` cela signifie `inv(A)*b`. C'est la division à gauche. Comment comprendre cela ? Rappelons nous que la division à droite (la division usuelle) était définit à l'aide de l'inverse : `A/B` signifie `A*inv(B)`.
La division à droite est identique. La matrice sous le signe division (\ ou /) est inversée et multipliée (en conservant l'ordre) avec la seconde matrice.

Pourquoi cela est il utile ? Tout simplement car cela étend le formalisme de la résolution des systèmes d'équations linéaires. Lorsque l'on a : $AX = b$, on peut alors dire que l'on a $X = A\backslash b$.

```
>> A = [3 5 1 ; 7 -2 4 ; -6 3 2]; b = [1 -3 3]';
X = A\b

X =
   -0.3100
    0.3886
   -0.0131
```

Exercice

(1.10) Celcius et Fahrenheit. Opérations matricielles
La relation entre les échelles de températures Celsius et Fahrenheit est linéaire. On peut la représenter par une relation de la forme

$$T_C = aT_F + b$$

où T_C est la température en degrés Celsius et T_F la température en degrés Fahrenheit.

Deux points sont bien connus sur ces deux échelles : le point de fusion de l'eau, de à $T_F = 32$ et $T_C = 0$ et le point d'ébullition à $T_F = 212$ et $T_C = 100$.
On peut donc identifier les coefficients a et b grâce à ces données.

a. Donnez le système d'équation qui permet d'obtenir a et b.

b. Donnez la forme matricielle de ce système.

c. Résolvez ce système en utilisant l'inversion matricielle puis en utilisant la division à gauche.

d. Donnez une représentation graphique permettant de convertir entre les échelles de Celsius et de Fahrenheit.

1.7 Représentation graphique

Le but de ce cours est de vous donner les outils permettant d'utiliser Matlab pour rédiger des documents scientifiques. Une compétence importante est donc la capacité à réaliser des graphs de qualité présentant clairement vos données. Matlab n'est pas le langage le plus adapté pour faire de "jolis" graphs. Igor ou R seront des outils plus spécifiquement orientés vers la mise en forme. Cependant, les récentes versions de Matlab ont fait d'énormes progrès dans ce domaine et il est maintenant possible d'obtenir des rendus très correct avec Matlab.

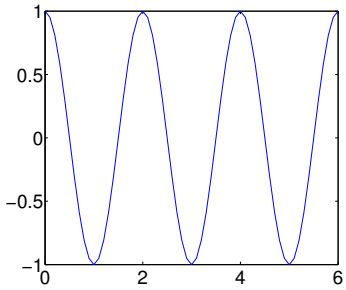


FIGURE 1.2: Représentation graphique sous Matlab de la fonction $f(x) = \cos \pi x$ entre 0 et 6.

1.7.1 Graphique d'une fonction

Le graph le plus simple est l'affichage d'une fonction d'une variable réelle. Voyons comment tracer $f(x) = \cos \pi x$.

```
>> X = [0:0.1:6]; Y = cos(X*pi);
figure(1);
plot(X,Y)
```

On présente sur la figure 1.2 le résultat obtenu dans Matlab. Détaillons le code. La commande `X = [0:0.1:6]` permet de d'échantillonner l'axe des absices. On utilisera des points espacés de 0.1 entre 0 et 6. Ensuite `Y = cos(X*pi)` défini le vecteur `Y` suivant la fonction $f(x) = \cos \pi x$. La commande `figure(1)` crée une nouvelle fenêtre sous Matlab nommée Figure 1. On utilisera par la suite si l'on souhaite conserver cette figure et en afficher une seconde `figure(2)`, etc. Enfin la commande `plot` est très simple d'usage : `plot(X,Y)`. L'aide de la fonction `plot` permet de connaitre les méthodes pour personnaliser son graph.

La commande `gca` renvoie le dernier axe selectionné (l'axe x par défaut).

Regardons ce que fait l'exemple suivant :

```
x = -pi:.1:pi;
y = sin(x);
plot(x,y)
set(gca,'XTick',-pi:pi/2:pi)
set(gca,'XTickLabel',{'-pi','-pi/2','0','pi/2','pi'})
title('Fonction Sinus');
xlabel('Radians');
ylabel('Valeur de la Fonction');
```

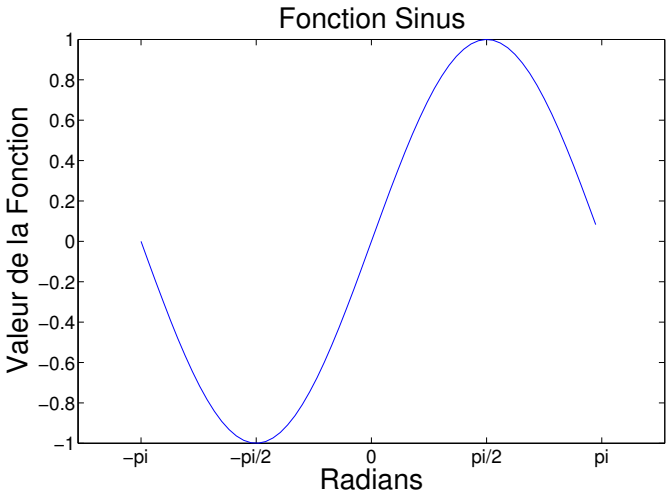


FIGURE 1.3: Exemple de graphique personalisé avec Matlab

Les deux premières lignes sont similaires à l'exemple précédent. Ensuite, on fixe les marques sur l'axe x et on assigne les labels. On place un titre sur la figure ainsi que sur les deux axes.

D'autres possibilités de mise en forme sont possibles. L'aide Matlab est la pour ca.

1.7.2 Graphique de plusieurs fonctions

Décrivons maintenant quelques fonctions très utiles pour les graphs. Si l'on souhaite placer deux courbes sur le même graph on utilise la commande `hold on` (à utiliser avec la commande `hold off`.)

```
close all;
x = -pi:.1:pi;
y = sin(x);
y2 = cos(x);
p1=plot(x,y);
hold on
p2=plot(x,y2);
set(p1,'Color','red','LineWidth',2)
set(p2,'LineWidth',2)

set(gca,'XTick',-pi:pi/2:pi)
set(gca,'XTickLabel',{'-pi','-pi/2','0','pi/2','pi'})
xlabel('Radians');
ylabel('Function Value');
```

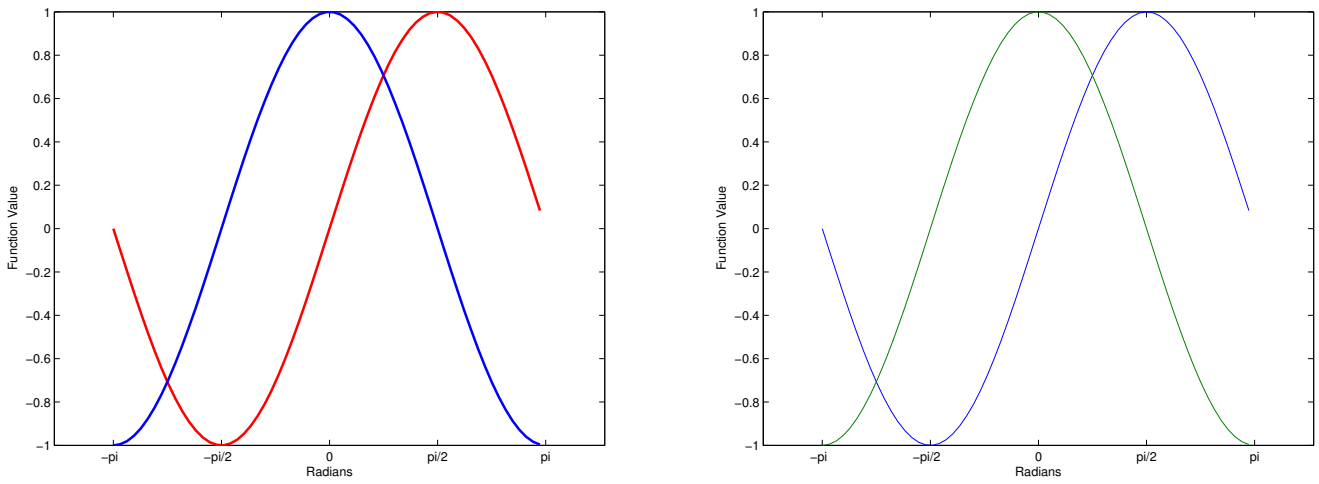


FIGURE 1.4: Deux exemples de courbes multiples sur un seul graph à l'aide de Matlab. La figure de gauche utilise le code ci-dessus, tandis que la figure de droite utilise le choix automatique des couleurs en remplaçant la commande `hold on` par `hold all`.

Une seconde méthode permet également d'afficher plusieurs courbes sur la même figure. Il suffit d'appeler ces differentes courbes dans la même fonction `plot` en les séparant par des virgules. On ecrira par exemple :

```
figure(1);
 h = plot(x,sin(x),x,cos(x));
xlabel('Radians');
ylabel('Function Value');
```

Cette expression est parfaitement équivalente à la précédente, elle permet d'afficher sur le même graph, les fonctions sinus et cosinus.

Une autre approche consiste à créer des sous-figures et à afficher les différentes fonctions dans ces différentes sous-figures. La commande à utiliser dans ce cas est `subplot(m,n,p)`. Cela va créer une matrices de graphiques m par n. Le graph souhaité est ensuite positionné à l'emplacement p. Les sous-figures sont numérotés de 1 à mn de gauche à droite puis de haut en bas. Le code suivant permet de générer la Figure 1.5 .

```
close all;
x = -pi:.1:pi;
y0 = sin(x);
y1 = sin(x+pi/3);
y2 = sin(x+2*pi/3);
figure(1);
subplot(3,1,1),plot(x,y0);
subplot(3,1,2),plot(x,y1);
subplot(3,1,3),plot(x,y2);
```

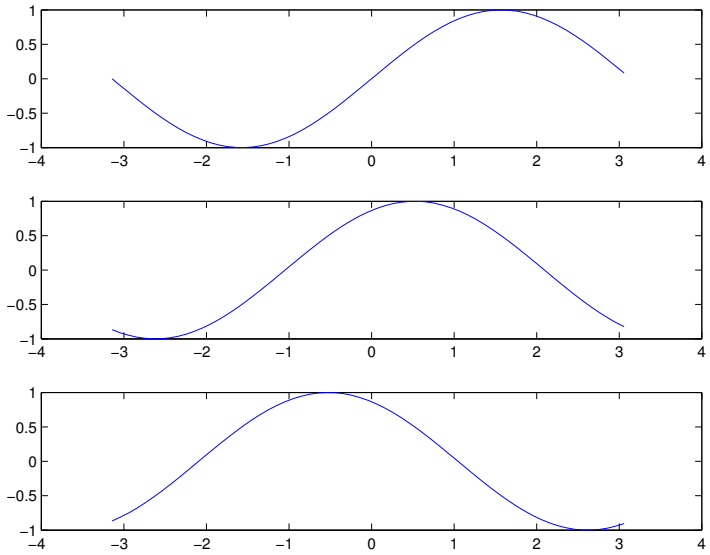


FIGURE 1.5: Démonstration de la fonction `subplot(m,n,p)`

Exercice

(1.11) Un exemple pas à pas

Pour mémoire les fonctions de Bessel sont les solutions canoniques de l'équation :
$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$, où α est l'ordre de la fonction solution.

a. Préparez vos données.
Nous souhaitons faire un graph contenant les fonctions de Bessel d'ordre 1,2 et 3 (`besselj(ordre,variable)` entre -10 et +10.
Créez une variable x qui varie entre -10 et 10 par intervalle de 0.1 et trois variables $y1, y2, y3$ étant respectivement la fonction de Bessel d'ordre n sur l'intervalle.

b. Sélectionnez une fenêtre et la position du graphique dans la fenêtre.
On souhaite afficher 4 graphiques. Un pour chaque fonction $y1, y2, y3$ et un ou l'on présente les 3 fonctions superposées. A l'aide de la commande `figure`, créez une nouvelle figure. Puis à l'aide de la commande `subplot`, disposez les 4 graphiques dans la fenêtre. Sur la première ligne, on place de gauche à droite les fonctions $y1, y2, y3$. Sur la seconde ligne, on place sur toute la largeur un graphique avec les 3 fonctions superposées. Lorsque vous utilisez la commande `plot`, n'oubliez pas de donner un nom à chacun de vos graphs : `p1=plot(...)` etc.

c. Choisissez votre type de ligne et de marqueur.
Pour fixer des paramètres, on utilise la commande `set(objet,'Parametre',valeur)`.
Voila une liste non exhaustive des paramètres :

`LineWidth` : largeur de la ligne. La valeur est un nombre réel (en point)

`LineStyle` : type de ligne. La valeur est un code par exemple : '--' pour pointillés.'

`Color` : couleur de la ligne. La valeur est soit un code 'r' pour rouge soit un triplet [rouge, vert, bleu]

Fixez la largeur du trait des trois graphiques de la première ligne à 1.5. Fixez la largeur du trait du graphique de la deuxième ligne à 2. La couleur des courbes sur le graphique de la deuxième ligne à 2 est fixée. Appliquez ces mêmes couleurs pour les trois graphiques de la première ligne. Enfin pour différencier les courbes en N/B, il est souvent utile d'utiliser différents types de ligne. Choisissez et appliquez 3 différents types de ligne pour le graphique de la deuxième ligne.

d. Fixez la limite des axes et la grille.
Encore un peu de mise en forme. La commande `grid on`, ajoute une grille sur le graphique de son choix. Ajoutez une grille sur le graphique de la deuxième ligne. On fixe ensuite les limites sur les axes des abscisses et des ordonnées à l'aide de la commande `axis([xmin xmax ymin ymax])`. On souhaite fixer les axes de la première ligne à $-10 \leq x \leq 10$ et $-1 \leq y \leq 1$. Sur la seconde ligne on zoome sur la partie centrale et on fixe $-5 \leq x \leq 5$ et $-1 \leq y \leq 1$.
On peut également raffiner et spécifier quelles marques on souhaite sur les axes (voir l'exemple précédent sur la fonction sinus).

On peut rappeler un graphique à l'aide de la commande `subplot`.

e. Ajoutez les légendes.
Bien que ce ne soit que le cinquième point de cette liste, l'ajout des légendes est fondamental ! Un graph sans légende est un graph sans intérêt. On fixera donc le nom des axes à l'aide des commandes `xlabel('nomX')`,`ylabel('nomY')`. Puis le titre des graphiques à l'aide de `title('titre')`, et enfin la légende des différentes courbes si nécessaires à l'aide de la commande `legend(h,'nom1','nom2','nom3')`.

f. Exportez vos données.
La dernière étape consiste à exporter vos données dans un format raisonnable (i.e. vectoriel !). Voila un exemple `print -dpdf -r600 test.pdf`, qui va sauvegarder en pdf avec une résolution de 600 ppi sur le fichier test.pdf.

Toutes ces actions peuvent être faites à l'aide de l'éditeur de graphs, mais le but de l'exercice ici est d'écrire un code Matalb qui va générer automatiquement cette figure. Au passage pour un traitement reproductibles des données, il est toujours conseillé de créer ses figures en script comme ceci.

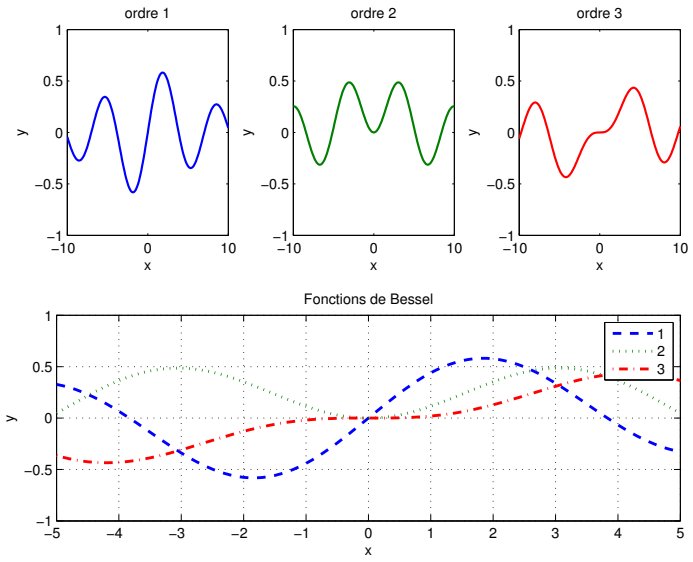


FIGURE 1.6: Résultat espéré de l'exercice 1.11

1.7.3 Bar graphs et Box plots

Nous allons discuter maintenant d'autres types de graphiques, plus adapté à la présentation de données de type liste. Pour ce faire je vais utiliser un exemple. Nous avons quatre listes de valeurs qui correspondent aux notes d'étudiants en français, maths, physique et anglais. On souhaite faire un graph synthétique qui permet d'évaluer les résultats de la classe dans les différentes matières. On peut utiliser le *box-plot* à l'aide de la fonction `boxplot(M)`, où `M` est une matrice qui comprend les notes des élèves organisées par colonne (colonne 1 : français etc..). La commande sera simplement :

Sur ces graphiques on peut lire plusieurs grandeurs statistiques. Le chapitre 2 sera consacré à l'etude détaillée de ces grandeurs. Notons ici simplement que la ligne rouge est la médiane, les limites des boites sont les 25 et 75 pourcentiles, les traits pointillés permettent de donner la répartition de l'ensemble des points, en ommétant les valeurs abbérantes (comme le 1 en anglais) qui sont représentées indépendamment.

```
>> boxplot(M,'labels',{'francais','anglais','physique','maths'});
```

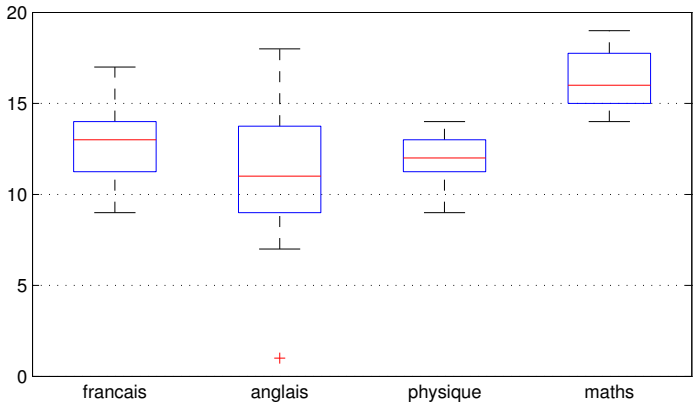


FIGURE 1.7: Box-plot représentant les notes des éleves en fonctions des matières.

Cette commande permet aussi de faire des boxplots lorsque l'on dispose de série de données organisées en vecteurs. Par exemple un vecteur `Origin` qui liste l'origine d'une voiture et un vecteur `MPG` qui liste le nombre de miles par gallon (unité US...) que peut parcourir cette voiture. On utilisera alors la commande

```
>> load carsmall; %pour charger les données.
>> boxplot(MPG,Origin);
```

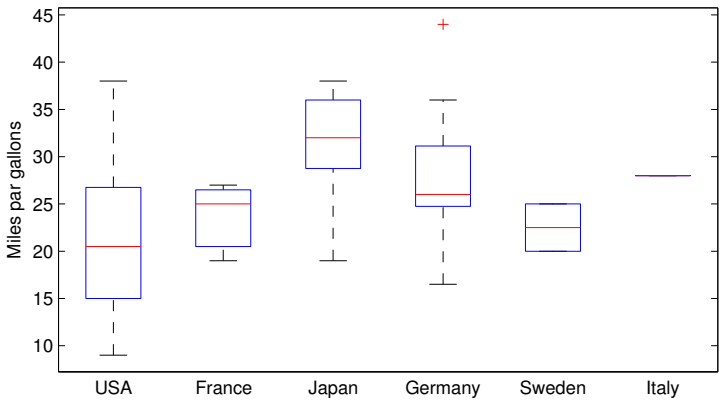


FIGURE 1.8: Box-plot représentant le nombre de miles par gallon en fonction du pays d'origine.

Un autre graphique classique de l'analyse de données est le bar-graph. Il associe une barre verticale (ou horizontale) de taille proportionnelle à la quantité mesurée

en fonction d'une variable descriptive (le mois de l'année, l'heure de la journée, le poids etc..) Par exemple si l'on souhaite décrire le taux d'ensoleillement et la quantité de pluie en fonction du mois de l'année on obtient les graphique suivants :

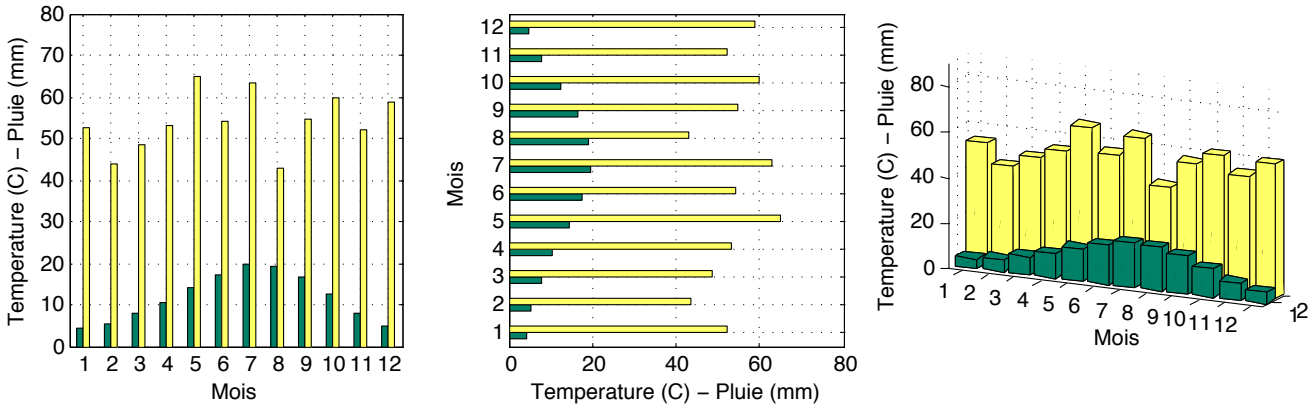


FIGURE 1.9: Bar plots en utilisant les fonctions `bar, barh, bar3`.

1.7.4 Histogrammes

Dans le même registre on peut aussi réaliser des histogrammes. Lorsque l'on dispose d'une série de données (reprenons les notes des étudiants par exemple), on peut faire un histogramme pour donner la répartition dans les différentes *zones* de notes. Par zone on entend par exemple entre 0 et 5 puis entre 5 et 10, 10 et 15, et 15 et 20. On peut également raffiner et avoir des zones par point (voir par demi-point).

Ici je présente un exemple d'histogramme de l'espérance de vie par habitant en fonction du pays d'origine.

On peut aussi normaliser l'histogramme grâce à l'astuce suivante :
`h = hist(D,100);`
`h = h/sum(h); bar(h);` , où D sont les données et 100 est le nombre de zones.

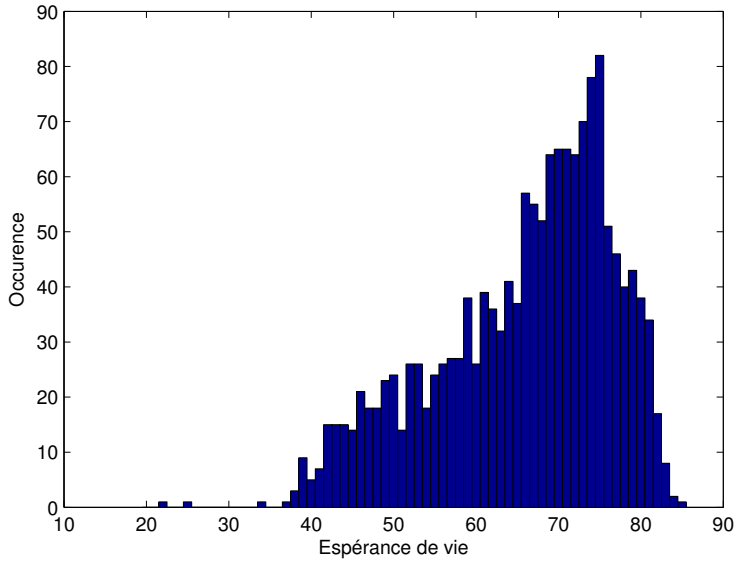


FIGURE 1.10: Exemple d'histogramme de l'espérance de vie par habitant en fonction du pays d'origine

1.7.5 Nuage de points

Il existe une grande variété de graphiques disponibles. Pour plus de details je conseille la documentation Matlab disponible ici :
`http://courses.washington.edu/css485/graphg.pdf`.

Je présente un dernier type de graph avant de passer aux fonctions de deux variables. Il s'agit du nuage de points, utiles pour présenter des données expérimentales. C'est la même commande que pour les fonctions : la commande `plot`, mais avec un option supplémentaire :

On peut utiliser la même option lorsque l'on trace des points expérimentaux et que l'on ne souhaite pas qu'une courbe les relie.

```
>> plot(x,y,'r+')
```

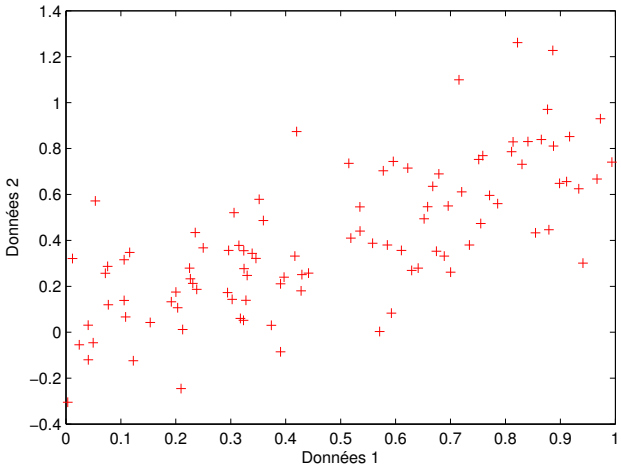


FIGURE 1.11: Exemple de nuage de points. On utilise l'option '+' pour obtenir des croix au lieux des lignes entre les points.

1.7.6 Graphiques des fonctions de 2 variables

Les graphs en des fonctions de deux variables sont légèrement plus compliqués à manier que ceux à une variable. En effet, de la même façon que l'on avait créer un vecteur qui discrétisait l'axe des abscisses, on doit créer un maillage qui discrétise le plan : c'est le but de la fonction `meshgrid`.

```
x = -10:0.1:10;
y = -5:0.1:5;
[X,Y] = meshgrid(x,y);
```

Dans cet exemple on crée 2 vecteurs x et y allant de -10 à $+10$ et de -5 à $+5$ respectivement, puis un maillage `[X,Y]` correspondant. A l'aide de ce maillage, on peut alors créer une variable g, qui va prendre la valeur de la fonction souhaitée (ici une Gaussienne) :

```
g = exp(-X.^2).*exp(-Y.^2);
```

On appelle ensuite une des fonctions d'affichage 3D, par exemple :

```
figure(1)
surf(x,y,g)
```

ou

```
mesh(x,y,g)
```

ou encore

```
pcolor(x,y,g)
```

et enfin

```
contour(x,y,g)
```

Si l'on souhaite rendre le graphique plus 'joli', on peut supprimer le maillage à l'affichage grâce à la commande `shading interp`.

Dans la figure suivante je présente ces différents rendus :

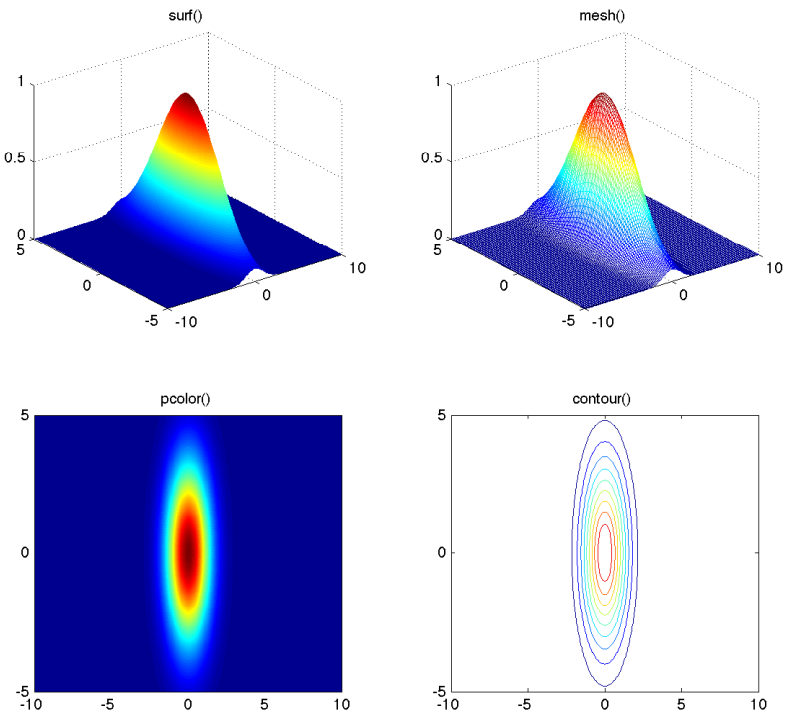


FIGURE 1.12: Différents rendus avec les fonctions `surf`,`mesh`,`pcolor`,`contour`

Enfin si l'on dispose d'une matrice qui représente pour chaque point la valeur du point on peut faire le graphique directement en utilisant la fonction `contourf`.

```
contourf(peaks(20),10);
colormap hot
```

Cet exemple utilise la fonction prédéfinie `peaks(n)` pour créer une matrice de taille 20x20 avec des valeurs correspondant à cette fonction. Le second argument de la fonction `contourf` est le nombre contour souhaité. Enfin la fonction `colormap` permet de changer le code couleur de arc-en-ciel (par défaut comme dans la figure 1.12) vers `hot` dans le cas présent.

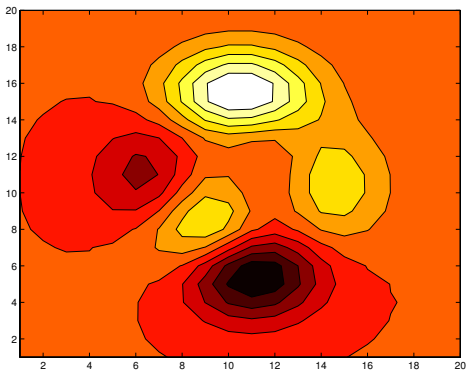


FIGURE 1.13: Fonction `contourf` en utilisant `colormap hot`

1.7.7 Images

Une image 640 pixels horizontaux par 480 pixels verticaux, où chaque pixel a trois couches : (rouge, vert et bleu) chacune codées sur 8 bits (0 à 255) fera donc une taille $T = 1280 * 960 * 3 * 8 = 29\ 10^9$ bits de données brutes. On verra dans les chapitres suivants comment on peut compresser ces images.

Un des grands domaines d'application du calcul matriciel est le traitement d'image. Une image est constitué par un ensemble de pixels. Chaque pixel est une (ou plusieurs) valeurs dans une matrice. Une image en niveaux de gris utilisera une valeur par pixel. Une image en couleur utilisera par exemple 3 valeurs : une pour le rouge, une pour le vert, une pour le bleu.

Pour afficher une image sous Matlab, la première étape est d'importer les données en utilisant `imread`. Cette commande permet de créer une matrice (pouvant avoir plusieurs couches). Ensuite on peut afficher cette image avec la commande `image()` ou `imagesc()` si l'on veut maintenir les dimensions de l'image originale.

```
>> A=imread('photo.tif','tiff');
>> image(A)
```

Pour sauvegarder une image on utilise la commande `imwrite()`.

```
>> imwrite(A,'test.jpg','jpg')
```

Normalement Matlab utilise des nombres à virgule flottante codés sur 64 bits. Il peut parfois il y avoir des problèmes avec les images et le format de données utilisés. On se referrera au manuel pour ces questions.

Enfin la commande `mat2gray` est utile pour convertir une matrices de données en image en niveaux de gris.

Exercices

(1.12) Fonctions de deux variables
Il est bien connu que la diffraction d'une onde électromagnétique de longueur d'onde λ par une fente rectangulaire donnera une intensité dans le plan d'observation en champ lointain en

$$I(x,y) = I_0.\mathrm{sinc}^2\left(x\frac{a}{\lambda d}\right)\mathrm{sinc}^2\left(y\frac{b}{\lambda d}\right), \tag{1.1}$$

où a est la dimension de la fente selon l'axe x et b est la dimension de la fente selon l'axe y.

(1.13) Images

1.8 Notions de programmation

La dernière section de ce chapitre d'introduction porte sur l'utilisation des structures de programmation. On l'a déjà vu, les scripts et les fonctions sont deux outils incontournables de Matlab. Cependant pour faire des scripts complexes il est souvent nécessaire de faire appel aux structures de programmation comme les boucles et les réalisations conditionnelles.

1.8.1 Boucles

Une boucle permet de répéter la même commande un grand nombre de fois en faisant varier un paramètre. C'est par exemple très utile pour la définition des suites par récurrences comme nous le verrons. Différentes synthaxes existent. La boucle `while` s'écrit de la manière suivante :

```
n=0;
while (n < 10)
   n.^2
   n=n+1;
end
```

Cette boucle, très simple affichera la valeur de n^2, pour tous les entiers n de 0 à 9. Une boucle se compose donc d'un compteur (ici n) initialisé à une valeur quelconque (ici 0), d'une condition de continuation (ici $x < 10$), d'une ou plusieurs commandes (ici 'affiche la valeur de n^2') et d'un incrément (ici incrément unité $n = n + 1$).

Cette même boucle peut également être réalisée avec la fonction `for` :

```
for n = 0:9
    n.^2
end
```

On retrouve les mêmes éléments : initialisation du compteur, continuation, commande à exécuter, et incrémentation (implicite).

1.8.2 Test SI

La structure conditionnelle permet d'effectuer une action si et seulement si un test préalable renvoie la valeur 'VRAI'. Dans sa version la plus complète la structure `if` permet aussi de prévoir ce que l'on fait si le test renvoie la valeur 'FAUX'

```
if (x > 0)
   x
else
   -x
end
```

Cette structure va retourner la valeur absolue de x. En effet si la condition $x > 0$ est vraie alors, on affiche la valeur x et sinon (`else`) on affiche la valeur $-x$. On peut complexifier ces structures en imbriquant des boucles dans des structures conditionnelles (ou l'inverse). Pour ne pas se perdre on veillera a bien respecter l'indentation du texte : c'est à dire de décaler d'une tabulation vers la droite à chaque fois que l'on imbrique une nouvelle structure.

Enfin une variante de la structure `if` est la structure `switch` qui permet de choisir parmi non pas VRAI et FAUX mais de multiples options appelés `case`. Par exemple

```
mynumber = input('Tapez un chiffre parmi -1,0 ou 1:');

switch mynumber
    case -1
        disp(' c est moins un');
    case 0
        disp('c est la valeur zéro !');
    case 1
        disp('c est plus un');
    otherwise
        disp('Une autre valeur');
end
```

Ce script va afficher le message 'c est moins un' si l'on tape -1, 'c est la valeur zéro !' si l'on tape 0 et 'c est plus un' si l'on tape $+1$. Si la valeur entrée n'est ni -1,0 ou 1 alors Matlab renvoie : 'Une autre valeur'.

Exercices

(1.14) Suite de Fibonacci

a. La suite de Fibonacci est définie de la manière suivante : $a_1 = 1$; $a_2 = 1$; $a_{n+2} = a_n + a_{n+1}$ pour $n > 0$. A l'aide d'une boucle `for`, calculez le vecteur `vec` de dimension 1x50 tel que `vec`$(n) = a_n$.

b. Que vaut a_{50} ?

c. Rédigez un programme « Fibo.m » qui permette a pour paramètre d'entrée un entier n (quelconque) et qui renvoie en sortie a_n. C'est-à-dire : en tapant `K = Fibo(n)` on obtient $K = a_n$.)

d. Que vaut a_{122} ?

e. Combien de temps prend ce calcul ?

f. Créez le vecteur `B` de dimension 1x49 tel que `B(n)` $= \frac{a_{n+1}}{a_n}$.

g. Tracez `B(n)` en fonction de n.

h. Pour n tendant vers l'infini, la limite de B s'appelle le *nombre d'or*. Donner une valeur approchée de ce nombre.

(1.15) Calcul d'une factorielle

a. A l'aide d'une boucle `while`, calculez 170!. Pour mémoire factoriel de 170 est défini par $170! = 1 \times 2 \times \ldots \times 170$.
b. A l'aide d'une boucle `for`, calculez 170!.

c. Quelle est la technique la plus rapide ? Comparez au temps mis par la fonction de Matlab `factorial()`.

b. Rédigez un programme « Facto.m » qui calcule le factoriel d'un nombre.

c. Qu'obtient-on pour 200 ! ?

Travaux Pratiques

Problème 1 : Partie de dé

On souhaite simuler un jeu où des joueurs lancent une poignée de 5 dés et cherchent à réaliser :
- un brelan (3 dés identiques) rapportant 20 points
- un carré (4 dés identiques) rapportant 40 points
- une main (5 dés identiques) rapportant 70 points
- une suite (1-2-3-4-5 ou 2-3-4-5-6) rapportant 60 points.

Nous utiliserons la commande `rand` qui permet de générer des nombres aléatoires compris entre 0 et 1, distribués avec une probabilité uniforme entre 0 et 1 (c'est-à-dire que tous les nombres entre 0 et 1 ont une probabilité égale d'être tirés). On peut noter qu'il s'agit en fait de nombres « pseudo-aléatoires », générés l'un après l'autre par Matlab à l'aide d'un algorithme tel que chaque nombre paraisse totalement indépendant du précédent, et donc aléatoire ; cependant, le résultat du calcul est parfaitement reproductible (ce qui est pratique pour moi lorsque je corrige vos TPs). Ainsi il est essentiel de fixer ce que l'on appelle la graine, c'est à dire la base de l'algorithme. Pour ce faire vous DEVEZ lancer la commande `rng(0)` au début de votre programme. Les nombres aléatoires que vous allez générer seront les mêmes que ceux de votre voisin-e.

Question 1.1
On souhaite à titre préliminaire simuler un tirage à pile ou face. Rédigez un programme `Alea2.m` qui ne demande aucune variable d'entrée et fournisse, comme variable de sortie, une quantité A qui vaille aléatoirement 0 ou 1 (ces deux valeurs ayant chacune la probabilité $\frac{1}{2}$.
On pourra générer tout d'abord une quantité `x = rand`, puis si $x < \frac{1}{2}$ on posera $A = 0$, et si $x > \frac{1}{2}$ on posera $A = 1$.

Question 1.2
On souhaite maintenant simuler le lancer d'un dé. Rédigez de même un programme `Alea6.m` dont la variable de sortie A vaille, aléatoirement et avec des probabilités $\frac{1}{6}$, 1, 2, 3, 4, 5 ou 6. On aura intérêt, pour cela, à lire attentivement l'aide de la commande `if` et à utiliser une succession de `elseif` ou bien d'utiliser la commande `switch`.

Question 1.3
A l'aide d'une boucle `for`, faites tourner 1000 fois le programme `Alea6`. Tracez l'histogramme des 1000 valeurs obtenues (commande `hist`). Vérifiez que l'on a bien approximativement tiré chaque face du dé le même nombre de fois.

Question 1.4
Rédigez un programme `Points.m` qui reçoive en entrée le résultat d'un lancer de 5 dés (la variable d'entrée est donc un vecteur M de dimensions 5x1 dont chaque élément est compris entre 1 et 6) et qui donne en sortie le nombre de points P obtenu par le joueur.

La méthode conseillée pour ce programme est de tracer l'histogramme H des valeurs de M (`H = hist(M,(1:6))`) puis de considérer le maximum de H. Il faudra donc, pour ce programme, rédigez une suite de conditions pour calculer P à partir de H donné, lui-même calculé à partir de M.

Question 1.5
A l'aide d'une boucle for, simuler 1000 lancers aléatoires d'une poignée de 5 dés (c'est-à-dire qu'à chaque lancer M = [Alea6 ; Alea6 ; Alea6 ; Alea6 ; Alea6]) et déterminer le nombre de points obtenus.
Quel est, sur ces 1000 lancers, la moyenne du nombre de points obtenus ? Le résultat du meilleur lancer ? Combien de fois la main (70 points) a-t-elle été obtenue ? Présentez ces résultats de manière synthétique à l'aide d'un box-plot.

Problème 2 : Colorimétrie

La colorimétrie est la science de la couleur et de la lumière. Dans ce problème exemple nous allons déterminer les composantes trichromatiques X, Y, Z de la lumière du jour transmise par un filtre donné à partir du spectre de transmission mesuré pour ce filtre et du spectre standard de la lumière du jour.

Préliminaires
Pour commencer il faut charger le fichier colorimetrie.mat sur le serveur : `http://www.quentinglorieux.fr/matlab/files/colorimetrie.mat`.
Ce fichier contient le vecteur `D65`, de dimensions 80x1, qui représente le spectre D65, caractéristique de la lumière du jour par temps couvert (en unités arbitraires), défini par la Commission Internationale de l'Eclairage (CIE). Les valeurs de ce spectre sont données pour des valeurs de longueurs d'onde allant de 380 à 775 nm, de 5 en 5 nm. Dans le fichier on obtient aussi la transmission $T(\lambda)$ du filtre (vecteur T de dimensions 93x1) mesurée pour des valeurs de λ espacées de 4,3 nm entre 380 et 775,6 nm.

Question 3.1
Créez un vecteur $L1$ de dimensions 80x1 contenant les valeurs 380, 385, 390, 395... 775, et tracez le spectre D65 en fonction de la longueur d'onde, en n'oubliant pas d'indiquer les axes et le titre.

Question 3.2
Créez un vecteur $L2$ de dimensions 93x1 contenant les valeurs 380, 384,3, 388,6... 775,6 et tracez $T(\lambda)$. Quelles longueurs d'onde sont coupées par le filtre ?

Question 3.3
On souhaite tracer la courbe du spectre de la lumière du jour transmise par le filtre soir : D65(λ)*T(λ. Une difficulté est que la transmission a été mesurée pour des valeurs de λ (vecteur $L2$) différentes de celles (vecteur $L1$) pour lesquelles D65 est donnée par la CIE, si bien que les deux matrices D65 et T ne peuvent pas être directement multipliées. Par une interpolation, créez un vecteur $T2$ de dimensions 80x1 contenant les valeurs de la fonction T(λ) pour les valeurs de λ contenues dans le vecteur $L1$. Tracez D65(λ)*T(λ).
Note : la commande permettant de faire une interpolation est : `interp1(L2,T,(380:5:780),'linear') ;`

Question 3.4
La CIE a aussi défini les courbes de référence $x(\lambda)$, $y(\lambda)$ et $z(\lambda)$, données dans les vecteur X,Y et Z de dimensions 80x1, pour des valeurs de λ espacées de 5 nm de 380 à 775 nm. Ces courbes permettent, pour un spectre S donné, de définir ses composantes trichromatiques X, Y et Z par :

$$X = \int S(\lambda)x(\lambda)\,\mathrm{d}\lambda, \qquad Y = \int S(\lambda)y(\lambda)\,\mathrm{d}\lambda, \qquad Z = \int S(\lambda)z(\lambda)\,\mathrm{d}\lambda.$$

Calculer les coordonnées trichromatiques (X,Y,Z) des spectres de la lumière du jour (S = D65) et de la lumière du jour transmise par le filtre (S = D65*T).

Question 3.5
La CIE définit la fonction linéaire multivariables qui permet de passer des composantes (R,G,B) aux composantes (X,Y,Z) :

$$\begin{cases} X = & 2.7689R + 1.7517G + 1.1302B, \\ Y = & 1.0000R + 4.5907G + 0.0601B, . \\ Z = & 0.0565G + 5.5943B \end{cases}$$

a. Quelle est la matrice P permettant de passer des composantes (R,G,B) aux composantes (X,Y,Z) (c'est-à-dire que [X ;Y ;Z] = P*[R ;G ;B]) ?

b. Quelle est la matrice permettant de faire l'opération inverse : passer des composantes (X,Y,Z) aux composantes (R,G,B) ?

c. Quelles sont les coordonnées (R,G,B) de la lumière du jour ? de la lumière du jour transmise par le filtre ? Ces coordonnées sont-elles cohérentes avec les couleurs que l'on peut observer quand on regarde ce filtre (qui devrait circuler dans la classe) ?

d. En déduire pour chacun de ces cas les coordonnées chromatiques (r,g,b) données par : $r = \frac{R}{R+G+B}$; $g = \frac{G}{R+G+B}$ et $b = \frac{B}{R+G+B}$.

Problème 3 : Augmentation de la résistance électrique avec la température

Chaque conducteur électrique possède une certaine résistance R. Cette résistance est directement proportionnelle à la longueur L du matériau et inversement proportionnelle à la surface de sa section A. On obtient :

$$R = \rho \frac{L}{A}, \tag{1.2}$$

où l'on appelle le facteur de proportionnalité ρ : la résistivité. La résistivité du cuivre à 20°C est de $\rho_{20} = 0.0168\ \Omega\text{mm}^2/\text{m}$. La résistivité est fonction de la température :

$$\rho(\theta) = \rho_{20}[1 + \alpha(\theta - 20)], \tag{1.3}$$

où $\rho(\theta)$ est la résistivité en fonction de la temperature θ en Celcius. α est un coefficient de température (pour le cuivre $\alpha = 0.0068\ \text{K}^{-1}$).

Une formule différente a été proposée par Wieseman en 1989 :
Wieseman (1989) donne une relation plus détaillée :

$$\rho(\theta) = \rho_{20}[1 + \alpha_W(\theta - 20) + \beta_W^2(\theta - 20)], \tag{1.4}$$

où $\rho_{20} = 0.0168\ \Omega\text{mm}^2/\text{m}$, $\alpha_W = 4.3\ 10^{-3}\ \text{K}^{-1}$ et $\beta_W = 0.6\ 10^{-6}\ \text{K}^{-1}$. On souhaite comparer graphiquement les deux relations, entre 20°C et 100°C.

Question 3.1
Créez un tableau de températures θ allant de 20°C à 100°C avec un pas de 0.5°C pour la formule (1.3) et (1.4).

Question 3.2
Tracer les deux résultats sur le même graphe Mettez les labels nécessaires aux axes avec les commandes `xlabel, ylabel`. Mettez une légende aux courbes en utilisant la commande `legend`.

Question 3.3
Imaginons maintenant un câble sous-marin en cuivre traversant la Manche. La largeur de la Manche varie entre 30 km au niveau du détroit de Dover à 320 km à son endroit le plus large. Un câble typique fait 69 mm de diamètre. Quel est la résistance d'un tel câble en fonction de sa longueur ? Présentez vos résultats sous la forme d'un graphe.

Question 3.4
Bien sur ces câbles sont protégés et isolés de le température de l'eau. Cependant un changement d'un degré de la température de la Manche induit un changement de 0.1 degré du cable (10 %). Durant l'année la température de l'eau varie entre 11°C et 21°C. Tracez la variation de de la résistance en fonction de la température et de la longueur du câble en utilisant la relation (1.4).

Question 3.5
Voici la température en fonction du mois de l'année pour 2012. Janvier : 11.3°C, Février 12.4°C, Mars 12.5°C, Avril 13.2°C, Mai 14.7°C, Juin 16.8°C, Juillet 17.8°C, Aout 19.1°C, Septembre 20.4°C, Octobre 17.6°C, Novembre 14.3°C, Décembre 12.2°C.
Tracez maintenant la variation de de la résistance en fonction du mois de l'année et de la longueur du câble en utilisant la relation (1.4).

Chapitre 2

Traitement du signal

2.1 Séries de Fourier

Avant de rentrer dans le vif sujet et d'étudier la transformée de Fourier, nous allons nous arrêter un instant sur les séries de Fourier. Ces dernières permettent de comprendre les concepts qui sous-tendent toute l'analyse de Fourier pour le traitement du signal que nous verrons dans ce cours et sont, selon moi, plus intuitives pour aborder ces questions.

Définition

Pour une fonction $f(x)$ définie sur $[-\pi, \pi]$, on peut écrire sa décomposition en série de Fourier de la façon suivante :

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nx) + b_n \sin(nx). \tag{2.1}$$

Nous allons maintenant utiliser Matlab pour obtenir les coefficients a_n et b_n.

Exercice 2.1

1. Ecrire $\int_{-\pi}^{\pi} f(x)\cos(mx)\,\mathrm{d}x$ à l'aide de l'équation (2.1).
2. Décomposer l'expression précédente et isoler les 3 termes :

$$P_1 = \frac{a_0}{2}\int_{-\pi}^{\pi} \cos(mx)\,\mathrm{d}x,$$

$$P_2 = \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi} \cos(nx)\cos(mx)\,\mathrm{d}x,$$

$$P_3 = \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi} \sin(nx)\cos(mx)\,\mathrm{d}x.$$

3.a Utiliser une représentation graphique sous Matlab pour trouver la valeur de P1.
3.b Utiliser une représentation graphique sous Matlab pour trouver la valeur de P2.
3.c Utiliser une représentation graphique sous Matlab pour trouver la valeur de P3 pour $n \neq m$.
3.d Utiliser une représentation graphique sous Matlab pour trouver la valeur de P3 pour $n = m$.

4. Déduire de ce calcul la valeur des coefficients a_n et monter que $a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)\,\mathrm{d}x$, pour tout n.

5. Effectuer la même démarche pour obtenir la valeur des coefficients b_n et monter que $b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)\,\mathrm{d}x$, pour tout n.

Notation exponentielle

Dans l'exercice précédent nous venons de voir que l'on peut décomposer une fonction définie sur $[-\pi, \pi]$ à l'aide de fonctions sinus et cosinus. Il est possible de faire cette décomposition de manière plus compacte à l'aide de la notation exponentielle. On écrira alors $f(x)$ définie sur $[-\pi, \pi]$ sous la forme :

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\pi x}. \tag{2.2}$$

Notons que l'on peut étendre à des fonctions $f(x)$ définie sur $[-L, L]$ par :

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{n\pi}{L}x}. \tag{2.3}$$

Exercice 2.2

En utilisant la relation d'Euler : $e^{i\theta} = \cos\theta + i\sin\theta$, démontrer que :

a. $c_n = \frac{a_n - ib_n}{2}$ pour $n > 0$
b. $c_n = \frac{a_n + ib_n}{2}$ pour $n < 0$.
c. $c_0 = \frac{a_0}{2}$.

2.2 Application à la synthèse de signaux sous Matlab

Très bien, nous avons donc maintenant compris que les séries de Fourier correspondent à des décompositions d'un signal sur la *base* des fonctions trigonométriques. Essayons de faire un exemple sous Matlab.

Exercice 2.3 : Effet la somme non-infinie sur les séries de Fourier

1. Créer un script qui va réaliser les opération suivantes :

   - Définir la taille L de l'échantillon utilisé.
   - Créer un vecteur `time` $100L$ points régulièrement espacés entre 0 et L.
   - Créer un vecteur `signal` contenant $100L$ points valant tous 0.
   - Définir a=[0 0 0 0 0 0] ; et b=[0 0 0 0 0 0] ;

2. Les vecteurs a et b sont les 6 premières composantes de la série de Fourier. Ajouter au script une boucle permettant de générer une série de Fourier tronquées au 6 premiers coefficients (équation 2.1).

3. Choisir les vecteurs `a` et `b` au hasard et afficher le signal en fonction de time.

4. Essayer les vecteurs $a_n = 0$ et $b_n = (-1)^{n+1}\frac{2}{\pi n}$ et afficher le signal en fonction de time. Qu'en dites vous ?

5. Ajouter maintenant 2 termes dans a et b (toujours $a_n = 0$ et $b_n = (-1)^{n+1}\frac{2}{\pi n}$) et afficher le signal en fonction de time. A-t-il changé ? Pourquoi ?

6. Réaliser une boucle pour ajouter un nombre arbitraire de coefficients dans la série et faire un test avec $a_n = 0$ et $b_n = (-1)^{n+1}\frac{2}{\pi n}$. A partir de combien de coefficients l'approximation d'un signal triangulaire vous semble bonne ?

7. Trouver la décomposition de Fourier d'une fonction *créneau* et réaliser un script afin de voir l'effet du nombre de coefficients sur la qualité de l'approximation.

Exercice 2.4 : Synthé Matlab

Dans cet exercice nous allons réaliser un synthétiseur Matlab ! Pour ce faire nous allons procéder comme à l'exercice précédent en définissant les composantes fréquentielles à ajouter dans notre signal. Puis nous utiliserons la fonction `sound` pour jouer ce magnifique morceau de musique électronique... C'est la première fois que nous allons développer un petit projet Matlab qui utilisera plusieurs fonctions, soyez attentifs à vos noms de fichier.

1. Créer une fonction qui prendre en paramètres d'entrée : T la durée du signal généré et $freq$ la fréquence de la composante. Cette fonction donnera en sortie un vecteur qui sera le signal généré.

2. La fréquence d'échantillonnage de Matlab par défaut pour le son est 8192 points par seconde. On définit donc une variable `time` telle que cette variable soit un vecteur régulièrement espacé entre 0 et T avec une longueur 8192*T. On crée aussi une variable `signal` de même taille et de valeur 0.

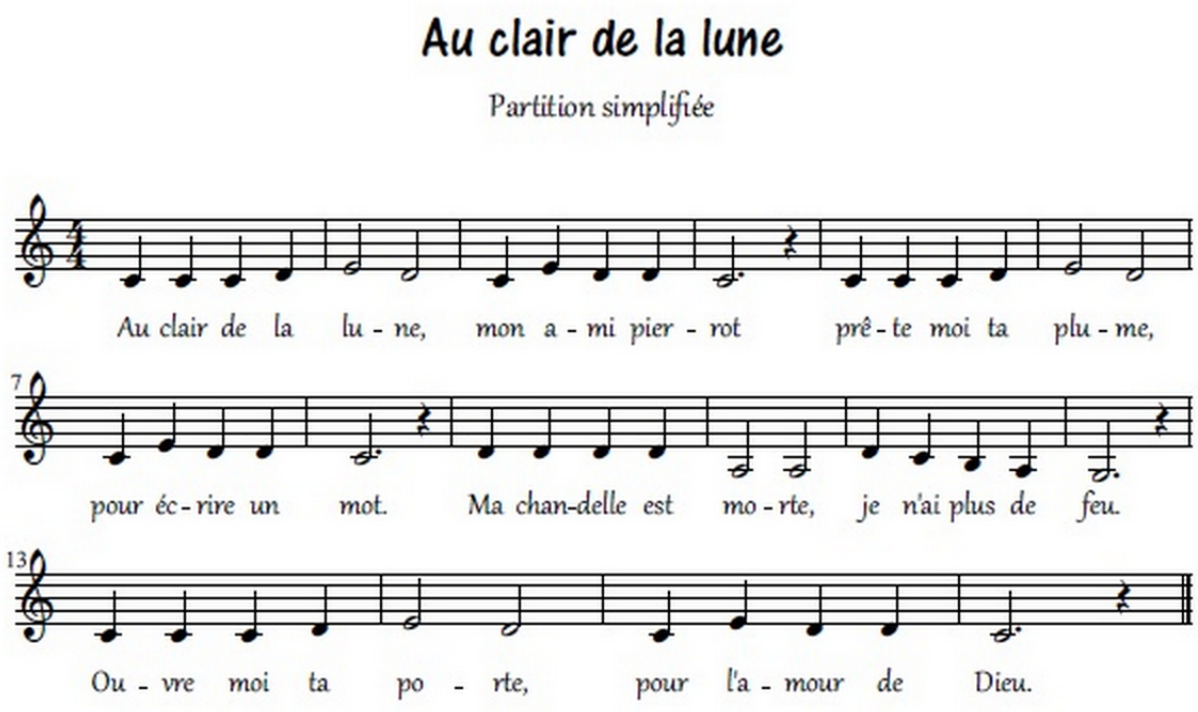
3. Créer un vecteur a de dimension égale à la fréquence maximale $freq$ et dont tous les coefficients valent 0 sauf le coefficient $freq$ qui vaut 1.

4. Réaliser une boucle qui permet de faire la synthèse du signal (equation 2.1). Attention pour avoir les fréquences en Hz et le temps en seconde, il est important d'avoir ici l'expression $\cos(n * time * 2\pi)$ dans la série de Fourier.

5. Tester le signal avec la commande `sound(...)`. PAS plus long que 2 secondes s'il vous plait !

6. Modifier le programme pour qu'il prenne en entrée non pas une valeur de fréquence mais un vecteur contenant plusieurs fréquences.

7. Faire un script qui appelle votre fonction afin de jouer "Au clair de la lune".



2.3 Transformée de Fourier

2.3.1 Définition

Nous venons de voir qu'il était possible de décomposer (ou synthétiser c'est la même chose), la plupart des signaux analytiques à l'aide des séries de Fourier. Il s'agit d'une somme des différents coefficients a_n et b_n si l'on utilise la forme trigonométrique ou c_n si l'on utilise la forme exponentielle. Lorsque l'on passe du discret (somme) au continu (intégrale), on dit que l'on réalise une transformée de Fourier. Il faut bien comprendre que l'on parle ici presque du même animal mathématiques, la principale différences étant que l'on ne réalise plus une somme des différentes composantes. Par analogie avec la relation (2.2), on écrit pour une fonction f sa décomposition de Fourier :

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k)\, \mathrm{d}k. \tag{2.4}$$

Les composantes de Fourier se retrouvent donc dans la fonction $F(k)$. Pour obtenir les coefficients de la décomposition, c'est à la fonction $F(k)$, on fait l'opération :

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x)\, \mathrm{d}x. \tag{2.5}$$

C'est cette opération qui s'appelle la transformée de Fourier. La transformée de Fourier possède de nombreuses propriétés très intéressantes, et je vous suggère de passer quelque temps avec un bon livre de Mathématiques pour les revoir...

2.3.2 Transformée de Fourier discrète

On vient de voir que pour passer de la série de Fourier à la transformée de Fourier on passait du discret au continu. Or, vous le savez maintenant, Matlab (et les ordinateurs en général) travaillent avec des données discrètes ! Il faut donc utiliser utiliser un algorithme pour faire l'opération "transformée de Fourier discrète".

La bonne nouvelle est que Cooley et Tukey dans le milieu des années 60 ont trouvé un algorithme très efficace pour faire des transformée de Fourier qui s'appelle FFT (Fast Fourier Transform). Cette algorithme repose sur une division du problème en 2 sous-problèmes, eux même divisés en 2 sous-sous-problèmes, etc. Il faut donc un nombre de point en 2^N pour que l'algorithme FFT s'applique bien, ce qui est une condition importante lorsque vous travaillerez avec la commande Matlab `fft`. Voyons maintenant comment utilise-t-on cette commande.

2.3.3 Un exemple pas à pas de FFT

Réalisons ensemble le code suivant :

```
clear all; close all; clc;

L=20;
n=128;
x2=linspace(-L/2,L/2,n+1); x=x2(1:n);

u=exp(-x.^2);
plot(x,u)

ut=fft(u)
plot(ut) % sur certaine version de Matlab il faut faire plot(real(ut)).
```

Ajouter maintenant la commande `fftshift` :

```
uts=fftshift(ut)
plot(abs(uts))
```

Il faut maintenant redéfinir correctement l'axe des fréquences :

```
k=(2*pi/L)[0:n/2-1 -n/2:-1];
```

`2pi/L` permet de normaliser au domaine L et non pas au domaine 2pi.

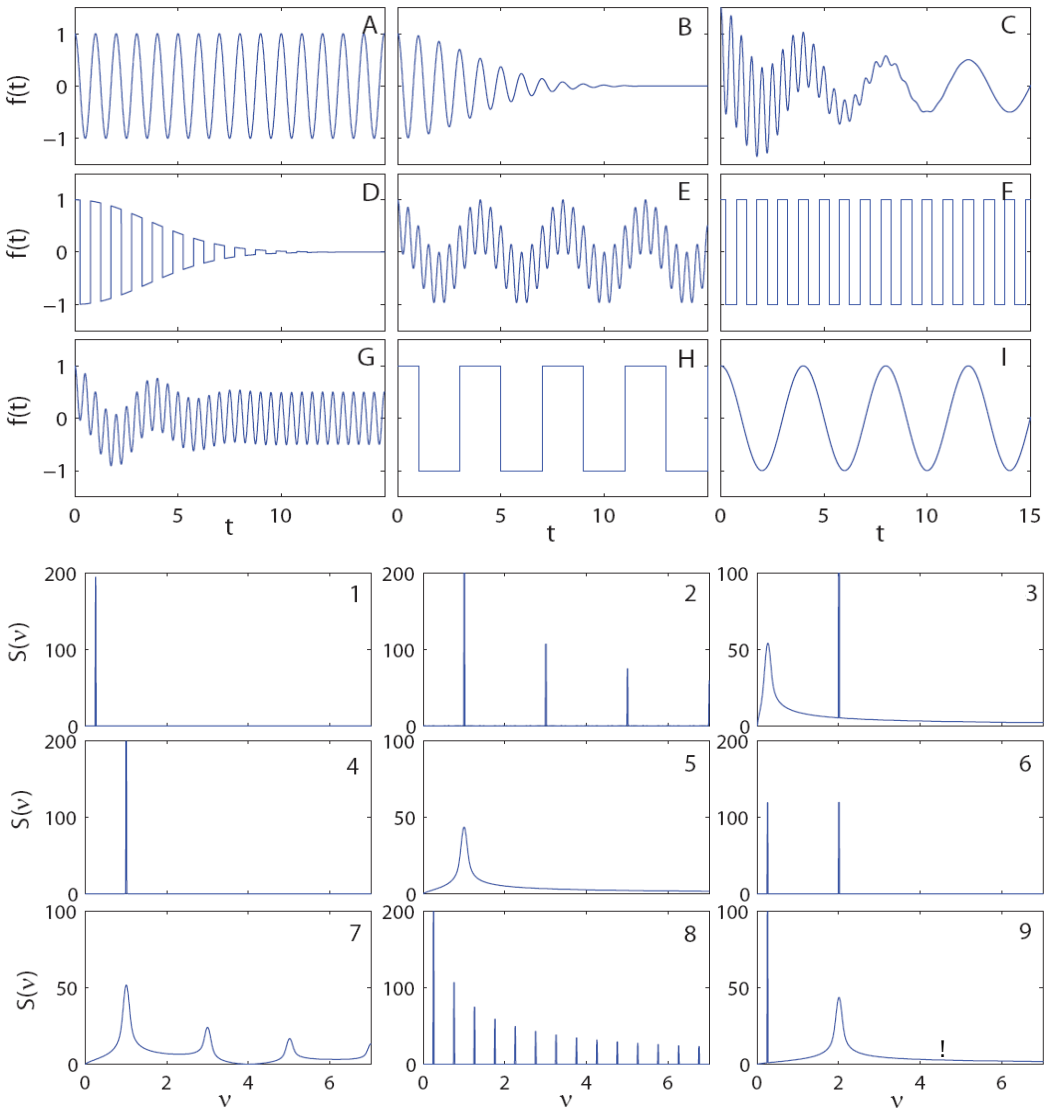
Exercice 2.6 : FFT d'un signal electrique - Rapport signal sur bruit.

On a mesuré le signal électrique $U(t)$ (tension en fonction du temps) avec un échantillonnage temporel $dt = 0.1$ ms pour des valeurs de t comprises entre 0 et 10 s. La tension $U(t)$ mesurée est donnée dans la matrice U de dimensions (100001,1). Elle contient une composante périodique et un bruit aléatoire.
Télécharger ce fichier depuis `http://quentinglorieux.fr/matlab/files/U.mat`

1.Tracer $U(t)$.
2.Tracer le spectre $S(f)$, c'est à dire le carré de la norme de la transformée de Fourier, exprimé en fonction de la fréquence f pour la fonction $U(t)$.
3. Quelle est la période de ce signal ?
4. Calculer le rapport signal sur bruit. Ce rapport est défini comme $\frac{I_1}{I_2 - I_1}$, avec I_1 l'intégrale de S pour f entre 39.9 et 40.1 Hz et I_2 l'intégrale de S pour f entre 0 et 5000 Hz.

Exercice 2.7 : Identification des transformée de Fourier

Associer chaque fonction A, B, C... à sa transformée de Fourier 1, 2, 3 ... en justifiant par les propriétés de la fonction.
Exemple : G – 3 La fonction G est une somme d'une sinusoïde de période 4 amortie sur un temps caractéristique de 5 et d'une sinusoïde de période 0,5. Le spectre 3 : un pic de fréquence 0,25 de largeur environ 0,2 et un pic étroit de fréquence 2.



Problème 2.8 : Filtrage sonore

Nous allons analyser spectralement des enregistrements sonores. On pourra créer, pour gagner du temps, la fonction TFourier.m suivante :

```
function [Nu,F] = TFourier(f,dt)
N = max(size(f));                % On détermine le nombre de points de la fonction f(t)
Nu = (0:N-1)/(N*dt);             % Rappel : Ttotal est donné par N*dt
F = abs(fft(f));                 % On prend la valeur absolue de la TF.
```

On peut ouvrir un fichier .wav dans Matlab en utilisant l'icône « ouvrir » de l'espace de travail (ou avec la commande wavplay). Ce fichier consiste en une matrice à une ligne, correspondant à un signal $S(t)$ enregistré à la fréquence d'échantillonnage `freq` c'est-à-dire que $S(t)$ est connu pour des valeurs de t séparées de $1/freq$. On peut l'écouter à la fréquence d'échantillonnage `freq` par la commande `wavplay(W,freq)`. Par défaut (si on tape `wavplay(W)`) la fréquence d'échantillonnage est fixée à la valeur standard `freq` = 11025 Hz.
Télécharger les fichiers « flute.wav » et « violon.wav » depuis `http://quentinglorieux.fr/matlab/files/`

Question 1.
Une onde sonore sinusoïdale de fréquence 784 Hz correspond à la note Sol. Ouvrir les fichiers « flute.wav » et « violon.wav », où la note Sol, jouée par un instrument, est enregistrée à la fréquence d'échantillonnage 10000 Hz.
a- Combien de temps dure chaque enregistrement ?
b- Tracer la transformée de Fourier de chaque enregistrement, en faisant attention à l'axe des abscisses. Décrire ces courbes : pic principal à 784 Hz, harmoniques, dédoublement des pics, pics parasites à basse fréquence etc.
c- Tracer la courbe de l'enregistrement de flûte entre 250 et 350 ms. Comment les différentes caractéristiques observées sur la transformée de Fourier se retrouvent-elles sur le signal ?

Conclusion : c'est notamment la présence d'harmoniques qui fait la différence, pour une même note (une même fréquence), entre deux instruments.

Question 2.
a- Par quelle commande peut-on générer artificiellement un signal sonore Wsol d'une durée de 2 secondes, à la fréquence d'échantillonnage 11025 Hz, de la note Sol ?

b- Tracer la transformée de Fourier de ce signal.

c- Si une note correspond à une fréquence donnée (par exemple, Sol : 784 Hz), la fréquence double correspond à la même note mais à l'octave suivante. Qu'obtient-on si on écoute le signal sonore Wsol à la fréquence d'échantillonnage 22050 Hz ? et à la fréquence 5512 Hz ?

Question 3.
On souhaite filtrer les hautes fréquences du signal « flute ». Pour cela, on calcule la transformée de Fourier, on supprime la partie contenant les fréquences hautes, puis on effectue la transformée de Fourier inverse pour obtenir le signal filtré :

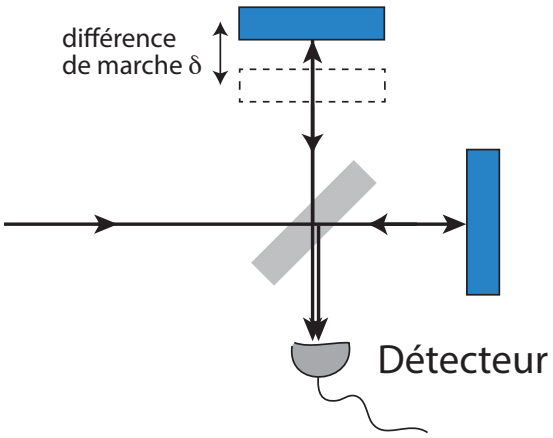
```
F = fft(Wflute); F(300:7200)=0; Wfiltre = real(ifft(F));
```

Commentez le code ci-dessus puis tracer le signal Wfiltre obtenu et sa transformée de Fourier. Montrer que l'on a réussi à supprimer la composante à 800 Hz et à faire ressortir la composante à 150 Hz.

Question 4.
On souhaite générer artificiellement la note de musique jouée par un violon. Pour cela, repérer sur le spectre du « Sol » du violon la hauteur des harmoniques 1 à 6, que l'on notera A_1 à A_6, et créer pour f = 784 Hz le signal $A_1\cos(2\pi ft) + A_2\cos(4\pi ft) + A_3\cos(6\pi ft) + A_4\cos(8\pi ft) + ...$
Faire de même pour la flûte. Ecouter ces deux signaux : a-t-on correctement reconstruit les sons des instruments ?

Problème 2.9 : Spectroscopie de Fourier (exercice d'examen 2010)

On cherche le spectre d'un signal lumineux. Ce spectre est noté $S(k)$ avec k le nombre d'onde défini comme $k = \frac{1}{\lambda}$ et exprimé en μm^{-1} .
On envoie le signal lumineux dans un interféromètre de Michelson (on rappelle son schéma ci-dessous à titre indicatif mais sa compréhension n'est pas nécessaire pour l'exercice). L'expérience consiste en la mesure de la puissance lumineuse P en sortie de l'interféromètre à l 'aide d'un détecteur (typiquement une photodiode) pour différentes valeurs de la différence de marche δ. On donne le résultat $\Delta P = P(\delta) - P(\delta = +\infty)$ de cette mesure (en unité arbitraire) dans la matrice `deltaP` en fonction de δ (en μm) dans la matrice `delta`. Les matrices sont à charger sur `http://quentinglorieux.fr/matlab/files/`.



Question 1.
Tracer $\Delta P(\delta)$ et placer les légendes.

Question 2.
La technique de spectroscopie par transformée de Fourier repose sur le fait que le spectre recherché $S(k)$ n'est autre que la norme de la transformée de Fourier de la fonction $\Delta P(\delta)$ mesurée. Tracer donc $|S(k)|$.

Question 3.
Combien cette courbe comporte-t-elle de pics ? En quelles longueurs d'onde sont-ils situés ? Lesquels vous paraissent significatifs, lesquels attribuez-vous à des artefacts ?

Question 4.
Quelle est la résolution sur k (c'est-à-dire l'échantillonnage dk entre deux valeurs de k successives) que nous pouvons atteindre dans ces conditions expérimentales ? Qu'aurait-il fallu changer dans l'expérience pour améliorer la résolution ?

Question 5.
On sait que :

$$\cos(k_1 x) + \cos(k_2 x) = 2\cos\frac{(k_1+k_2)x}{2}\cos(\frac{(k_1-k_2)x}{2}).$$

Quel lien peut-on faire entre la présence d'un « doublet » (deux pics très proches) dans la transformée de Fourier et la présence de « battements » (modulation des oscillations par une enveloppe sinusoïdale) de $\Delta P(\delta)$?