INSA Toulouse
Département STPI
PO ICBE 2ème année
UV3 d'informatique
2007-2008

Travaux pratiques de programmation Matlab

Arnaud Cockx

Fabrice Deluzet

Samy Gallego

Alain Huard

Alain Liné

Jean-Christophe Michelucci

Jérôme Morchain

Violaine Roussier-Michon

Table des matières

Chapitre 1

TP d'initiation (4 séances TP)

MATLAB est un logiciel de calcul et de visualisation, dont les entités de base sont des matrices : MATLAB est une abréviation de Matrix Laboratory.

MATLAB est un langage interprété : il propose des facilités de programmation et de visualisation, ainsi qu'un grand nombre de fonctions réalisant diverses méthodes numériques.

La meilleure façon d'apprendre à utiliser ce logiciel est de l'utiliser vous même, en faisant des essais, en commettant des erreurs et en essayant de comprendre les messages d"erreur qui vous seront renvoyés. Ces messages sont en anglais !

Ce document est destiné à vous aider pour quelques premiers pas avec MATLAB. Les sections et sous-sections signalées par **(*)** donnent des compléments qui pourront vous être utiles à l'avenir, mais peuvent être réservées pour une lecture ultérieure.

1.1 Acces à MATLAB

Les ordinateurs sur lesquels vous allez travailler sont configurés sous un environnement Linux. Si vous pouvez en général naviguer dans vos dossiers de manière assez naturelle comme sous Windows, il est important de connaître quelques commandes Unix qui vous permettent certains raccourcis.

Pour ouvrir une session, vous devez taper votre login et votre mot de passe. Si le login est unique pour chaque utilisateur, le mot de passe est modifiable ! Pour fermer votre session, allez dans le menu "démarrer" et choisissez "logout"ou "quittez l'environnement"(cf figure 1.1).

FIG. 1.1 – Quitter une session Linux

Pour travailler dans cet environnement Linux, soit vous naviguez avec votre souris de manière plus ou moins intuitive, soit vous tapez des commandes Unix dans un shell (ou fenêtre de commande en français). Pour ouvrir un shell, on parcourt le menu "démarrer", on choisit "System" et "Terminal (Konsole)" (figure 1.2). Pour quitter le shell il suffit de

taper “exit”.

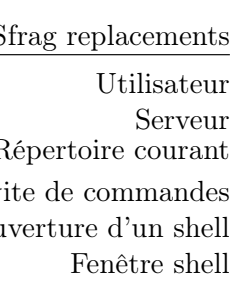

FIG. 1.2 – Ouverture d'une fenêtre shell (konsole).

Les quelques commandes Unix à connaitre sont :

1. `cd` : change le répertoire de travail
2. `ls` : donne le contenu d'un repertoire
3. `rm` : efface un fichier
4. `cp` : copie un fichier
5. `mv` : déplace un fichier
6. `mkdir` : crée un répertoire
7. `rmdir` : détruit un répertoite (vide)
8. `man` : donne le manuel d'aide d'une commande
9. `jobs` : affiche la liste des travaux en cours lancé depuis le shell courant
10. `kill`n : stoppe le travail n
11. `ps` : liste l'ensemble des processus/travaux actifs
12. `whoami` : retourne l'identifiant (*login*) de l'utilisateur

13. `grep` : recherche un mot dans une liste

Par exemple la commande `ps -Ao pid,user,comm | grep ‘whoami‘` retourne l'ensemble des processus que l'utilisateur a initié :

```
$ whoami
username
$ ps -Ao pid,user,comm | grep ‘whoami‘
 7562 username  konsole
 7563 username  konsole
 7569 username  bash
 7578 username  bash
 9776 username  matlab
 9777 username  bash
10182 username  matlab_helper
10084 username  MATLAB
```

Le symbole “|” appelé `pipe` ou tube de redirection permet d'envoyer le résultat de la commande `ps` à la commande `grep`. La commande `ps -Ao pid,user,comm | grep ‘whoami‘` recherche donc le mot `username` dans la liste de l'ensemble des processus et retroune les lignes où il apparaît.

Pour interrompre un programme en cours, on tape “Ctrl C” (dans le shell). Cela peut ne pas être suffisant, dans ce cas on pourra utiliser la commande `kill` suivie du *PID (Process IDentifier)*. Par exemple si l'on souhaite interrompre une session MATLAB on utilisera :

```
$ ps -Ao pid,user,comm | grep ‘whoami‘ | grep -i matlab
10084 username  MATLAB
10182 username  matlab_helper
$ kill -KILL 10084
$ ps -Ao pid,user,comm | grep ‘whoami‘ | grep -i matlab
$
```

Vous aurez parfois besoin d'écrire dans des fichiers indépendants des lignes de commandes. Il vous faudra alors un éditeur de texte. Nous vous proposons d'utiliser EMACS qui se lance par le menu "démarrer" et la rubrique "éditeurs de texte". On ouvre et on ferme un fichier grâce au menu "file" ou simplement en tapant la commande `emacs` dans un shell. Le presse papier (copier-coller) se trouve dans le menu "edit".

Enfin, pour accéder au logiciel MATLAB, vous pouvez utiliser le menu "démarrer" ou taper simplement la commande `matlab` dans un shell (figure 1.3). Suivant la version (plus ou moins complète, c'est à dire avec ou sans l'environnement `java`) que vous utiliserez, vous aurez tout à disposition dans le logiciel MATLAB (l'éditeur de MATLAB s'appelle `edit` et est disponible avec la version complète de la figure 1.3(a)) ou vous aurez besoin d'un éditeur de texte comme EMACS extérieur à MATLAB. **Nous vous conseillons fortement d'utiliser la version la plus légère de** MATLAB **et l'éditeur** EMACS **pour travailler en ICBE à l'INSA**. L'expérience montre que la version complète "plante" très régulièrement.

Vous reconnaitrez la fenêtre de travail MATLAB (figure 1.3) par le “*prompt*” :

```
»
```

(a) Version compléte : `matlab -jvm`. (b) Version allégée : `matlab -nojvm`.

FIG. 1.3 – Les différentes session de MATLAB.

Vous pourrez quitter MATLAB en tapant dans cette fenètre :

```
» quit
```

Dans la suite de cette séance, vous êtes invités à taper les instructions affichées dans ce texte après ce "prompt". Les instructions et variables MATLAB sont données dans une typographie de machine à écrire. Les opérations mathématiques et valeurs représentées par ces variables sont écrites en italiques.

1.2 Variables et Matrices

1.2.1 Matrices

Dans MATLAB, toutes les variables représentent des matrices. Par exemple, on multiplie deux matrices `a` et `b` par `a*b,` et le produit de deux scalaires s'écrit de la même façon : ils sont interprétés comme des matrices `1X1`. On peut définir une matrice dans MATLAB de plusieurs façons :

- par la liste de ses éléments,
- en la générant par une suite d'instructions et de fonctions,
- en la lisant dans un fichier extérieur.

Si l'on représente la touche `enter` de votre clavier par le symbole ↩, les instructions suivantes :

```
» A=[3 2 -2;-1 0 1;1 1 0] ↩
```

et

```
» A=[ ↩
 3 2 -2 ↩
-1 0 1 ↩
 1 1 0 ]  ↩
```

créent la même matrice `3X3`. Dans la suite, le symbole ↩ ne sera plus indiqué.

En ayant éxecuté l'une ou l'autre de ces commandes, vous aurez remarqué que MATLAB affiche la valeur de la variable `A` que vous venez de définir. Vous pouvez éviter l'affichage

en faisant suivre la commande d'un point-virgule. Ne la retapez pas pour le vérifier ; en utilisant la flèche ascendante de votre clavier ↑, vous pouvez rappeler la commande

```
>> A = [3 2 -2;-1 0 1;1 1 0]
```

et ajouter le point-virgule

```
>> A = [3 2 -2;-1 0 1;1 1 0] ;
```

MATLAB distingue les majuscules et les minuscules ; si vous tapez :

```
>> a
```

vous recevrez le message :

```
??? Undefined function or variable 'a'.
```

L'énumération des éléments d'une matrice ou d'un vecteur peut être implicite. Essayez par exemple :

```
>> m = [-3:3]
>> x = [1:-0.25:-1]
```

Les matrices `m` et `x` sont des matrices à une ligne (vecteurs-lignes). MATLAB vous fournit les dimensions d'une variable par la fonction `size` :

```
>> size(m)

ans =
     1     7
```

Si l'on veut un vecteur colonne, on peut taper

```
>> m = m'
```

ou

```
>> x = x(:)
```

L'apostrophe définit la transposition : `A'` représente la matrice A^T. Les deux points (`:`) imposent le format colonne.

On peut aussi créer directement un vecteur en séparant les composantes par un point-virgule :

```
>> b = [3 ; 0 ;  2 ] ; size(b)
```

Un cœfficient quelconque d'une matrice est référencé par ses indices de ligne et de colonne, dans cet ordre. Par exemple, l'instruction :

```
>> A(1,2)
```

renvoie la réponse :

```
ans =
        2
```

`ans` est le nom d'une variable qui reçoit le dernier résultat d'une instruction ou d'une suite d'instructions lorsqu'il n'a été affecté à aucune variable. On peut faire s'afficher la valeur d'une variable sans affichage de son nom ni de `ans`

```
>> disp(A(1,2))
```

Certaines fonctions de MATLAB créent des matrices particulières. Les quelques matrices suivantes sont parmi les plus couramment utilisées :

- `eye(n)` renvoie la matrice identité, habituellement notée I_n en mathématiques ; essayez `eye(4)`,
- `ones(n,m)` renvoie une matrice à n lignes et m colonnes dont tous les coefficients sont des 1,
- `zeros(n,m)` matrice à n lignes et m colonnes dont tous les cœfficients sont des 0 ; elle sert beaucoup pour les initialisations.
- `linspace(a,b,n)` crée une matrice ligne de n points régulièrement espacés sur l'intervalle $[a,\ b]$ (bornes comprises).
- `rand` crée une matrice aléatoire. Essayez `» B = rand(3,4)`. Dans quel intervalle sont choisis les coefficients de cette matrice ?

La commande `help` permet de recevoir des informations (en anglais !) sur une fonction ; essayez :

```
>> help linspace, >> help rand
```

1.2.2 Opérations usuelles

MATLAB permet d'effectuer les opérations usuelles en étendant leur définition aux matrices selon les règles de l'algèbre linéaire.

L'addition et la soustraction ne méritent aucun commentaire particulier, sinon que les tailles des matrices doivent être égales, essayez :

```
>> 3+5
>> ans -2
>> A + ones(3)
>> m + x
```

La dernière commande aura entrainé cette réponse cinglante :

```
??? Error using ==> +
Matrix dimensions must agree.
```

MATLAB autorise que les dimensions ne soient pas égales lorsqu'une des variables est un scalaire. `A + 1` fournit la même valeur que `A + ones(3)`.

La multiplication de deux variables sera possible si les matrices qu'elles représentent respectent les règles de concordance des dimensions :

```
>> 3*4
>> c = A*b
>> C = A*B
>> b'*c
>> b*c'
```

Comme les variables `A` et `B` représentent respectivement des matrices 3×3 et 3×4, la variable `C` représente une matrice 3×4, mais la commande `B*A` susciterait un message d'erreur de dimensions.

On peut aussi remarquer que le produit scalaire des vecteurs b et c s'obtient directement par `b'*c` , alors que `b*c'` définira une matrice 3×3 (de rang 1 !).

MATLAB autorise que les dimensions ne concordent pas lorsqu'une des variables représente un scalaire.

MATLAB distingue la division à gauche et la division à droite :

```
>> 3/4
ans =
    0.7500
>> 3\4
ans =
    1.3333
```

Ainsi cette division doit elle s'interpreter comme le produit par l'inverse de la variable située du côté vers lequel penche la barre : `3/4` représente $3 \times \frac{1}{4}$ alors que $3 \setminus 4$ représente $\frac{1}{3} \times 4$. Cette idée se généralise aux variables représentant des matrices :

```
>> x = A\b ;
>> A*x - b
```

Vous aurez compris que la variable `x` contient la solution du système linéaire $A\,x = b$. Comment interpréter le dernier résultat renvoyé par la machine ? Au passage, vous remarquerez que cette nouvelle **affectation** de la variable `x` écrase la précédente.

Quelle reflexion vous suggère le résultat de :

```
>> b/A
```

On peut se demander ce que représente la matrice M définie par

```
>> M = A\eye(3)
```

MATLAB autorise le produit et les divisions par un scalaire.

L'élévation à une puissance donnée est définie par le symbole `^` : essayez

```
>> 2^3
>> A^2
>> A^0
>> A^(-1)
```

Vous avez retrouvé la matrice M.

Les règles de priorité des opérations sont conformes à celles qui sont utilisées dans la plupart des langages de programmation. Elles sont données par le tableau suivant :

Niveau de priorité	Opération
1	puissance
2	multiplication et division
3	addition et soustraction

A niveau de priorité égal, les opérations sont effectuées de la gauche vers la droite. Les parenthèses vous permettent d'organiser les priorités dans vos calculs. Leur utilisation est recommandée pour éviter les ambiguïtés et risques d'erreurs.

```
>> 2 + 3^2
>> 2 + 3*3^2
>> 2 + (3*3)^2
>> (2 + 3*3)^2
>> A^2\A
>> A*A\A
```

1.2.3 Opérations sur les tableaux

Une matrice peut être vue comme un tableau de valeurs, indépendamment de l'algèbre linéaire. MATLAB permet de travailler avec les valeurs contenues dans un tableau. Pour l'addition et la soustraction, il n'y a pas de différence entre matrice et tableau de valeurs. Pour la multiplication, les divisions et l'élévation à une puissance, on fait précéder le symbole d'opération d'un point. Les tableaux doivent avoir la même taille :

```
>> A.^2
>> M.*A
>> b./c
>> C.*B
>> A.*B
>> A.^[2 1 2 ; 2 0 1 ; 1 1 1]
```

On remarquera que `A.^ 2` élève à la puissance 2 chaque élément de A. C'est un raccourci pour `A.^ (2*ones(3))`.

1.2.4 Manipulations sur les matrices

MATLAB permet de manipuler les matrices par blocs. Essayez par exemple :

```
C = [ones(3), rand(3,2) ; rand(2,3), eye(2)]
```

On peut ensuite en extraire la sous-matrice formée des trois dernières colonnes :

```
C1 = C(:,3:5);
```

et on retrouve `ones(3)` en faisant afficher `C(1 :3,1 :3)`.

MATLAB propose quelques fonctions qui facilitent certaines manipulations. La fonction `diag` extrait la diagonale d'une matrice, sous forme d'un vecteur, ou crée une matrice diagonale, à partir d'un vecteur ; essayez par exemple les instructions suivantes :

```
>> R = rand(5);
>> diag(R)
>> diag(ans)
>> diag(diag(R))
```

L'argument général de `diag` est double ; entrez :

```
>> help diag
```

Vous constatez que l'on peut ainsi extraire d'une matrice ses éléments d'une "parallèle" à la diagonale. Essayez :

```
>> diag(R,1)
>> diag(R,-2)
>> d = ones(4,1);
>> M = 2*eye(5)-(diag(d,1)+diag(d,-1))
```

`tril` renvoie la partie triangulaire inférieure d'une matrice, éléments diagonaux compris. Elle admet aussi plusieurs arguments

```
>> tril(R)
>> tril(R,1)
```

`triu` renvoie la partie triangulaire supérieure d'une matrice.
`reshape` reformate une matrice, c'est-à-dire change le nombre de lignes et de colonnes, en prenant les éléments colonne par colonne.
`repmat` crée une "grande" matrice en recopiant une matrice donnée selon le format fournit.

1.3 Boucles et tests

Les principales instructions de contrôle proposées par MATLAB sont `for, while` et `if` ; elles fonctionnent à peu près comme leurs équivalents dans les autres langages de programmation.

La boucle `for` doit respecter la syntaxe suivante :
`for` *compteur = expression*
    instructions
`end`

Generalement, *expression* est un vecteur de la forme `début :incrément :fin` et *compteur* prend successivement toutes les valeurs de *expression* pour éxecuter *instructions*. Par exemple, les instructions suivantes permettent de calculer une valeur approchée de e^x, pour $x = 10$, en utilisant l'approximation :

$$e^x \simeq \sum_{k=0}^{n} \frac{x^k}{k\,!} \tag{1.1}$$

```
>> x = 10;
>> n = 50;
>> s = 1;
>> terme = 1;
>> for k = 1:n, terme = x*terme/k ; s = s + terme ; end;
>> s
```

On pourra vérifier plus tard que la valeur obtenue, $s \simeq 2.2026 \times 10^4$ est assez proche de e^{10}.
Plusieurs boucles peuvent être emboîtées. Comme MATLAB est un language interprété, il faut essayer d'éviter les boucles quand c'est possible. Pour cela on peut utiliser des boucles implicites. Pour construire la matrice $H = (h_{i,j}) \in M_3(\mathbb{R})$ avec $h_{i,j} = \frac{1}{i+j-1}$, on peut utiliser les instructions suivantes :

```
>> n = 3;
>> H = zeros(3);
>> for i = 1:n
       for j = 1:n
           H(i,j) = 1/(i+j-1);
       end;
  end;
```

Noter que l'on a initialisé la variable `H`. C'est recommandé lorsque c'est possible. On peut aussi construire `H` de la façon suivante :

```
>> J = 1:n;
>> J = repmat(J,n,1);
>> I = J';
>> E = ones(n);
>> H = E./(I+J-E);
```

L'instruction `while` respecte la syntaxe suivante :
`while` *expression*
instructions
`end`
Les *instructions* seront éxécutées tant que *expression* sera vraie. Les instructions suivantes permettent de trouver approximativement le plus petit nombre positif représenté dans l'arithmétique utilisée par MATLAB :

```
>> x = 1 ;  while x>0 , xm = x; x = x/2; end; xm
```

L'instruction `if` respecte la syntaxe suivante :
`if` *expression*
    instructions
`end`
Les *instructions* ne seront éxécutées que si *expression* est vraie. Il est possible de proposer une alternative, en indiquant les instructions à éxecuter lorsque *expression* est fausse :
`if` *expression*
    instructions 1
`else`
    instructions 2
`end`

1.4 Fonctions de base

1.4.1 Fonctions scalaires

Ce sont les fonctions usuelles. Par exemple :

`sin`	`exp`	`abs`	`round`
`cos`	`log`	`sqrt`	`tanh`
`tan`	`rem`	`sign`	`acosh`

Ces fonctions font partie des fonctions élémentaires proposées par MATLAB. Pour en avoir la liste, vous pouvez taper :

```
>> help elfun
```

Comme la liste est longue, vous pouvez contrôler son défilement :

```
>> more on, help elfun, more off
```

Le défilement se fait ligne à ligne (en tapant une touche quelconque) ou par pages de 20 lignes (en tapant la barre d'espacements). Vous pouvez maintenant comparer la variable `s` calculée précédemment avec e^{10} ;

```
>> s - exp(10)
```

Quelle commande devrez-vous entrer pour obtenir la valeur absolue de l'erreur relative ?
Ces fonctions traitent les variables matricielles comme des tableaux de nombres ; si la variable `A` représente une matrice A, la variable `S = sin(A)` représentera une matrice S dont les coefficients sont $s_{i,j} = \sin(a_{i,j})$. Pour l'exponentielle et la racine carrée, qui possèdent une version "matricielle", MATLAB propose les fonctions `expm` et `sqrtm`.

Le cas des polynômes est un peu particulier : un polynôme peut être défini par ses cœfficients : on utilise dans ce cas une matrice ligne : par exemple, le polynôme $Q(x) = x^3 + 2x^2 - 3$ sera représenté par :

```
>> Q = [1 2 0 -3];
```

Sa valeur au point $x = 1.2$ sera fournie par :

```
>> polyval(Q,1.2)
```

Les racines du polynôme Q seront fournies par la fonction `roots`. A l'inverse, on peut aussi déterminer les coefficients d'un polynôme à partir de ses racines en utilisant la fonction `poly` :

```
>> r=[1 2 3]; K = poly(r)
>> polyval(K,r)
```

Ces instructions assignent à `K` la valeur `[1 -6 11 -6]` qui définit le polynôme $K(x) = x^3 - 6x^2 + 11x - 6$. Le polynôme ainsi calculé a toujours le cœfficient de son terme de plus fort degré égal à 1. C'est un polynôme unitaire.
Comme les fonctions usuelles `polyval` traite les matrices comme des tableaux, il faudra utiliser la fonction `polyvalm` pour évaluer un polynôme de matrice. Comparez `polyval(K,A)` et `polyvalm(K,A)`.

Certaines fonctions spéciales sont également proposées par Matlab : vous allez rencontrer les fonctions de Bessel de première espèce notées $J_\nu(x)$; elles sont données par la fonction `besselj` dont le format d'appel le plus courant est, pour le paramètre entier `nu` et la variable `x` : `y=besselj(nu,x)` ;
Vous pouvez retrouver l'information sur le format d'utilisation par :

```
>> help besselj
```

La liste de ces fonctions spéciales vous sera renvoyée par la commande :

```
>> more on, help specfun, more off
```

1.4.2 Fonctions vectorielles

Ces fonctions sont plutot destinées à agir sur des vecteurs, lignes ou colonnes. Citons par exemple :

`max`	`sum`	`mean`	`sort`
`min`	`prod`	`std`	

dont vous trouverez l'objet par la commande `help` lorsqu'il n'est pas évident. Elles agissent aussi sur les matrices, mais colonne par colonne. Essayez par exemple :

```
>> b = rand(3,3) ; c = max(b) ; d = max(max(b));
>> b,c,d
>> sort(c)
>> sort(b)
```

1.4.3 Fonctions matricielles

Citons en quelques-unes :

`eig` : valeurs et vecteurs propres d'une matrice,
`inv` : inverse,
`expm` : exponentielle de matrice,
`size` : dimensions de la matrice,
`norm` : norme (2 par défaut, mais aussi 1 ou ∞),
`rank` : rang.

Une liste un peu plus complète, mais non exhaustive, de ces fonctions vous est fournie en annexe. Une liste complète est fournie en réponse à la commande

```
>> more on, help matfun, more off
```

1.5 Graphiques

1.5.1 Visualisation des courbes en 2D

MATLAB permet de tracer facilement le graphe de fonctions scalaires. Cela se fait à l'aide de la fonction `plot`. Elle prend en argument des paires de vecteurs de même dimension. Voici par exemple le moyen d'obtenir le graphe de la fonction $\cos 3x$ sur l'intervalle $[0, 2\pi]$:

```
>> x = [0:0.01:2*pi] ; y =cos(3*x) ; plot(x,y);
```

Le format de base de la fonction `plot` est en fait le suivant :

```
plot(x,y,s) ;
```

où `x` contient les absisses, `y` les ordonnées, et `s` est une chaine de 1 à 3 caractères : `s='ct'`, respectivement couleur et tracé, qui peuvent être choisis dans le tableau suivant :

Couleur		**Tracé**	
`y`	jaune	`-`	trait continu (c)
`m`	magenta	`:`	pointillés (c)
`c`	cyan	`-.`	trait-point (c)
`r`	rouge	`-`	tirets (c)
`g`	vert	`+`	plus (d)
`b`	bleu	`o`	cercles (d)
`k`	noir	`*`	étoiles (d)
`w`	blanc	`x`	croix (d)

Les types de tracé précédés de (c) sont des tracés continus, avec une interpolation entre les points fournis, alors que pour les autres, ne sont représentés que les points (x_i, y_i). On peut combiner ces deux modes de tracé. On peut tracer les graphes de plusieurs fonctions simultanément. Essayez maintenant :

```
>> z = sin(2*x);plot(x,y,x,z);
```

On peut alors légender notre graphique de la façon suivante :

```
>> legend('cos(3x)','sin(2x)');
```

On peut également obtenir ces deux courbes par la succession de commandes :

```
>> plot(x,y);
>> hold on;
>> plot(x,z);
```

La seule différence est que cette fois, les deux graphes sont tracés avec la même couleur. `hold on` gère le graphique courant de façon que les commandes graphiques à suivre vont s'ajouter à ce graphique. `hold off` est la valeur par défaut : dans ce cas une nouvelle commande `plot` effacera le graphique existant.
Les axes sont définis automatiquement ; on peut choisir les bornes des coordonnées du graphe à l'aide de la fonction `axis` . Essayez par exemple :

```
>> axis([-1 5 -1.5 1.5]);
```

Pour la suite de notre travail, il est préférable que vous entriez maintenant la commande :

```
>> hold off;
```

Le format de la fonction `plot` permet bien sûr de représenter facilement des courbes paramétriques :

```
>> plot(y,z);
```

On obtient, pour les valeurs définies plus haut, une courbe connue sous le nom de courbe de Lissajoux. Les instructions suivantes vont réduire légèrement l'échelle, donner un titre et un nom aux axes, et faire apparaître une grille sur le fond :

```
>> axis([-1.1 1.1 -1.1 1.1])
>> title('Courbe de Lissajoux');
>> xlabel('x : axe des absisses');
>> ylabel('y : axe des ordonnees');
>> grid
```

Vous obtiendrez l'impression sur papier d'un graphe en utilisant la commande `print` qui enverra l'impression sur l'imprimante à laquelle est relié votre ordinateur. C'est la dernière de vos figures qui sera imprimée. (A utiliser bien sûr avec parcimonie !)

Exercice

On considère la fonction définie par

$$F(x) = 0.01 \exp x + 10 \cos x - 3\,x. \tag{1.2}$$

Tracer la courbe représentant cette fonction sur l'intervalle $[-1, 10]$, en utilisant ses valeurs aux points $x_k = -1 + \frac{k}{100}$, $0 \leq k \leq 1100$. Fixer la taille de la fenêtre de visualisation de façon que les absisses soient comprises entre -1 et 11 et les ordonnées entre -50 et 200. Compléter cette figure en y ajoutant les axes et un titre. Ces axes peuvent être tracés "à la main" en utilisant les fonctionnalités de la fenêtre graphique (voir les icônes). On peut aussi les obtenir en tapant par exemple :

```
>> hold on ;
>> plot([0,9],[0,0],9,0,'>')
>> plot([0,0],[-20,150],0,150,'^')
```

Quant au titre, vous devez pouvoir l'obtenir sans indication ...

1.5.2 Visualisation des courbes en 3D

La fonction qui permet de représenter une courbe dans l'espace $\mathbb{R}^3$ est `plot3`. Sa syntaxe est

```
plot3(x,y,z,s) ;
```

et cette fois `x`, `y` et `z` sont des vecteurs de même dimension contenant les trois coordonnées. Essayez :

```
>> theta = pi*[-4:0.04:4]; r = linspace(1,6,201);
>> x = r.*(1+cos(theta)); y = r.*sin(theta); z = r;
>> plot3(x,y,z,'k*'); grid;
```

1.5.3 Visualisation des surfaces

MATLAB permet de représenter des surfaces données en coordonnées cartésiennes ou sous forme paramétrique. La forme paramétrique peut s'écrire :

$$\begin{aligned} x &= x(s,t) \\ y &= y(s,t) \\ z &= z(s,t) \end{aligned}$$

où les paramètres s et t parcourent un certain domaine. Plusieurs fonctions permettent cette représentation, parmi lesquelles `mesh` représente un treillis, et `surf` une surface pleine. Elles ont le même format d'appel :

```
surf(X,Y,Z,C) ;
```

`X,Y` et `Z` sont des matrices de mêmes dimensions contenant les coordonnées de points de la surface. `C` permet de définir les couleurs et peut être omis. Nous allons représenter le cône d'équation $x^2 + y^2 - 2xz = 0$ en utilisant la représentation paramétrique donnée pour $(\theta,\, \rho) \in]-\pi,\, \pi[\times]\, 0,\, 12[$ par :

$$\begin{aligned} x &= \rho\,(1 + \cos\theta), \\ y &= \rho\,\sin\theta, \\ z &= \rho. \end{aligned}$$

On peut procéder de la façon suivante :

```
>> n = 31 ; theta = pi*[-n:2:n]/n ; r = linspace(0,12,n);
>> X = r'*(1+cos(theta)) ;
>> Y = r'*sin(theta) ;
>> Z = r'*ones(size(theta));
>> surf(X,Y,Z);
```

Il est important de remarquer que `X`, `Y` et `Z` sont des matrices de même taille !
La surface représentée laisse apparaître les traces d'une grille qui la recouvre. L'aspect de la surface est contrôlé par la variable `shading`, dont les valeurs possibles sont `faceted` (valeur par défaut), `flat` (supprime le treillis), et `interp` (adoucit les transitions). Essayez :

```
>> shading interp;
```

Les couleurs sont définies par une matrice `C` de même taille que celles des coordonnées, et par défaut, cette matrice est `C=Z`, de sorte que d'une certaine façon, la couleur est "proportionnelle" à l'altitude du point de la surface. Tout cela dépend du choix d'une palette graphique. Essayez :

```
>> C = rand(size(Z)) ; surf(X,Y,Z,C);
```

La palette de couleurs de la fenêtre de visualisation est définie par la variable `colormap`. Sa valeur par défault est `'jet'`. Essayez :

```
>> colormap('cool');
```

La liste des palettes disponibles est fournie avec celle de toutes les fonctions ou variables de contrôle d'un graphique 3D par la commande :

```
>> more on, help graph3d, more off
```

Examinons à présent le tracé d'une surface donnée par son équation cartésienne $z = f(x, y)$. En général, on dispose d'une famille de valeurs de x et d'une famille de valeurs de y stockées dans 2 vecteurs, respectivement `x` et `y`. On va devoir construire deux matrices `X` et `Y` telles que les colonnes de `X` et les lignes de `Y` soient constantes : ces matrices seront utilisees pour le calcul des valeurs de f. On peut utiliser la fonction `meshgrid` . Ainsi, pour représenter la surface d'équation $z = \exp(-x^2 - y^2)$, $-2 < x, y < 2$, on pourra procéder comme suit :

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = X .* exp(-X.^2 -Y.^2);
>> surf(Z);
```

Si on ne souhaite que le treillis, on remplace la fonction `surf` par `mesh`. Si l'on souhaite des isolignes, on utilise la fonction `contour3`. Par exemple :

```
>> v  = linspace(0,max(max(Z)),20); contour3(Z,v);
```

1.6 M-fichiers

Jusqu'à présent, nous avons travaillé en ligne sur la fenêtre MATLAB. Cette façon de travailler n'est pas optimale. Si on commet une erreur, on est obligé de retaper toute la ligne ! Pour palier à cet inconvenient, on peut utiliser un fichier d'instructions. MATLAB peut en effet exécuter une suite d'instructions stockées dans un fichier. On appellera ces fichiers des M-fichiers, et leur nom doit être suivi du suffixe `.m` ; vous vous habituerez vite à travailler sous MATLAB essentiellement par l'intermédiaire de tels fichiers. Nous vous demandons de le faire systématiquement. On va distinguer deux types de M-fichiers : les fichiers "scripts" et les fichiers de fonctions.

1.6.1 Fichiers "scripts" ou fichiers d'instructions

Un fichier script est un fichier contenant une suite d'instructions MATLAB directement éxécutables. Ces instructions sont éxécutées en utilisant les variables de l'espace de travail. La liste de ces variables est fournie en réponse à la commande

```
>> whos
```

Pour créer un tel fichier, il faut ouvrir une fenêtre d'édition. Selon l'installation de MATLAB, vous aurez ou non accès à l'éditeur deboggueur. Si c'est le cas, votre fenêtre de travail ressemble à la figure ci-dessous :

Vous pouvez alors utiliser les options du menu de cette fenêtre.

Dans le cas contraire, et si vous travaillez sous LINUX ou sous UNIX, nous vous conseillons `emacs`. Vous y accédez depuis votre fenêtre MATLAB par la commande :

```
! emacs &
```

Une fois que vous aurez écrit les instructions que vous voulez voir executées, vous allez sauvegarder le fichier sous un nom de votre choix, avec le suffixe `.m` . Vous le ferez en vous servant du menu de la fenêtre `emacs`.

Un exemple

Lorsque que l'on veut calculer la dérivée d'une fonction $f(x)$ au point x, on peut utiliser la relation :

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h}. \tag{1.3}$$

On va vérifier la pertinence de ce calcul en fonction des valeurs du pas h dans le cas de la fonction $f(x) = \sin(x)$, dont on calculera la valeur approchée de la dérivée au point $x = 1$. Comme la réponse est connue, $f'(1) = \cos(1)$, on pourra calculer l'erreur et la représenter graphiquement.
Editer un fichier que vous appellerez `Dernumsin.m` dans lequel :

- vous definirez la variable `x = 1 ;`
- vous définirez un vecteur contenant les pas, `pas = 10.^ (-[1 :16]) ;`
- vous calculerez les dérivées numériques de la fonction sinus au point x pour ces pas, `Dn = (sin(x+pas)-sin(x))./pas ;`
- vous calculerez l'erreur `Err = abs(Dn-cos(x)) ;`
- vous représenterez cette erreur en fonction du pas en coordonnées logarithmiques : `loglog(pas, Err) ;`

Une fois que ce fichier est sauvegardé, vous pouvez en demander l'éxécution :

```
>> Dernumsin
```

Vous pourrez alors comprendre pourquoi nous avons utilisé des coordonnées logarithmiques. Les variables qui sont créées par l'éxécution du fichier script sont visibles depuis l'espace de travail :

```
>> clear all
>> whos
>> Dernumsin
>> whos
```

L'emploi d'une majuscule comme première lettre de votre fichier vous garantit contre le risque d'utiliser le nom d'une fonction prédéfinie de MATLAB.

Exercice

Le but de cet exercice est d'écrire un fichier appelé `Deform.m` dont l'objet est de tracer un cercle et son image par un endomorphisme donné. On rappelle que le cercle $\mathcal{C}$ de centre O et de rayon 1 est donné par son équation paramétrique

$$\begin{aligned} x(t) &= \cos(t) \\ y(t) &= \sin(t)\,, \quad t \in [0, 2\pi] \end{aligned}$$

Soit un endomorphisme de R^2 représenté par la matrice A suivante

$$A = \begin{pmatrix} 0.8630 & 0.1638 \\ 0.1638 & 0.7325 \end{pmatrix}$$

Alors l'image de $\mathcal{C}$ par A est l'ensemble des points $(u(t), v(t))$ tels que

$$\begin{pmatrix} u(t) \\ v(t) \end{pmatrix} = A \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

Créer un fichier `Deform.m` qui contient les instructions suivantes :
- définir le paramètre t (utiliser la fonction `linspace`)
- calculer les $x(t)$ et $y(t)$
- tracer sur une figure le cercle $\mathcal{C}$, mettre une légende et des axes
- définir la matrice A
- définir le vecteur $(x(t), y(t))$
- calculer le vecteur $(u(t), v(t))$
- tracer sur la même figure que précédemment le vecteur $(u(t), v(t))$ ainsi obtenu

Après l'avoir sauvé, vous pouvez commander l'exécution de ce fichier par la commande

```
>> Deform
```

On vérifie que l'image de $\mathcal{C}$ est bien une ellipse. Quelle est la particularité de cette ellipse si on choisit la matrice A diagonale ?

1.6.2 Fichiers "functions"

L'exemple du fichier `Dernumsin` précédent n'est pas tout à fait satisfaisant : dans certains cas, il serait préférable de disposer d'un moyen de calculer une dérivée approchée pour une fonction quelconque, au point que l'on veut et pour le pas que l'on choisit. Cela peut se faire en créant une **fonction**. Avant d'éxaminer la possibilité de créer une fonction de plusieurs variables, intéressons nous à des fonctions plus simples.

Fonctions simples d'une variable

Nous avons déjà rencontré la fonction $F(x) = 0.01\, e^x + 10\, \cos x - 3\, x$. Il est possible de créer une fonction MATLAB qui calcule les valeurs de F. Cela peut se faire de deux façons :
- comme la fonction F peut se définir en une seule ligne, on peut le faire avec la fonction `inline`
  ```
  >> F1 = inline('0.01*exp(x)+10*cos(x)-3*x')
  ```
- de façon plus générale, on peut éditer un fichier que l'on va appeler `F2.m` qui devra commencer par le mot-clef `function` : vous pouvez le définir de la façon suivante
  ```
  function y = F2(x) ;
  y = 0.01*exp(x)+10*cos(x)-3*x;
  ```

La première définition est à utiliser lorsque l'on n'envisage pas d'utilisation ultérieure de cette fonction. La définition au moyen d'un fichier permet de la conserver. Cependant, ainsi qu'elle est rédigée, cette fonction a encore le défaut que son objet n'est pas immédiatement

lisible ; on risque de l'avoir oublié quand on aura besoin de la réutiliser. Il faut donc la commenter. Cela se fait sur des lignes qui commencent par le symbole % . Ce symbole transforme en commentaires tous les caractères qui le suivent dans la ligne courante. Entre autres commentaires, vous devrez toujours indiquer le format d'appel de la fonction. Voici comment nous vous suggèrons d'écrire ce fichier :

```
function y = F2(x) ;
% fichier fonction definissant la fonction F2(x) qui sert d'exemple
% au polycopie d'initiation a Matlab.
% L'appel se fait selon :
%                     >> y = F2(x);
y = 0.01*exp(x)+10*cos(x)-3*x;
```

Ces commentaires sont placés immédiatement après la première ligne (de déclaration). De cette façon, ils constitueront la réponse fournie par MATLAB à la commande :

```
>> help F2
```

Les variables internes d'un fichier fonction sont locales ; elles ne sont pas vues depuis l'espace de travail. Corollairement, les variables de l'espace de travail que vous souhaitez utiliser dans une fonction doivent être passées en argument. La commande `global` permet de définir des variables globales. Elles doivent être déclarées globales au début de la fonction **et** dans l'espace de travail.

Fonctions de plusieurs variables

On souhaite créer une fonction qui calcule la dérivée numérique d'une fonction passée en argument, en un ou plusieurs points pour une valeur donnée du pas. Le fichier `Dernum.m` réalise une telle fonction :

```
function Dn = Dernum(fonc,x,h);
% Cette fonction calcule la derivee "numerique" de la fonction
% fonc au(x) point(s) x pour le pas h.
% L'appel se fait selon :
%            Dn = Dernum(fonc, x, h)
% Les variables en entree sont :
%         - fonc est le nom de la fonction dont on veut calculer
%         la derivee,
%         - x est une valeur reelle ou un vecteur contenant
%         des valeurs reelles du domaine de definition de fonc,
%         - h est le pas du calcul.
xp = x + h;
y = feval(fonc,x);
yp = feval(fonc,xp);
Dn = (yp - y)/h;
```

Le passage d'une fonction en argument mérite quelques commentaires :

- si `fonc.m` est une fonction prédéfinie de MATLAB (par exemple `sin`) ou le nom d'un fichier de fonction (par exemple `F2`), on pourra écrire l'un ou l'autre des deux appels ci-dessous :
  `>> Dn = Dernum('F2', x, pas);`
  `>> Dn = Dernum(@F2, x, pas);`
- si `fonc` est le nom d'une fonction définie par `inline`, (par exemple `F1`), on devra formuler l'appel selon :
  `>> Dn = Dernum(F1, x, pas);`

Essayez de définir une fonction MATLAB qui calcule la fonction $F_{(a,b)}$ qui présente un "pic" peu marqué au point a et un pic plus marqué au point b :

$$F_{(a,b)}(x) = \frac{2}{(x-a)^2 + 0.01} + \frac{1}{(x-b)^2 + 0.002}.$$

Les paramètres a et b seront des variables de la fonction que l'appellera par exemple `Foncapics`, et la première ligne du fichier `Foncapics.m` sera :

```
function y = Foncapics(x,a,b);
```

On prendra soin de définir cette fonction de façon qu'elle puisse prendre en arguments les éléments d'un tableau (`./` et `.^`).

Arguments optionnels en entrée (*)

Le nombre de variables en entrée n'est pas nécessairement fixé. La fonction `Dernumbis` propose, si on le désire, de faire une représentation graphique de la dérivée calculée.

```
function Dn = Dernumbis(fonc,x,h,dessin);
% Cette fonction calcule la derivee "numerique" de la fonction
% fonc au(x) point(s) x pour le pas h.
% L'appel se fait selon :
%            Dn = Dernumbis(fonc, x, h)
%      ou    Dn = Dernumbis(fonc, x, h, dessin)
% Les variables en entree sont :
%         - fonc est le nom de la fonction dont on veut calculer
%         la derivee,
%         - x est une valeur reelle ou un vecteur contenant
%         des valeurs reelles du domaine de definition de fonc,
%         - h est le pas du calcul,
%         - dessin est une variable optionnelle ; si on
%         passe une valeur, on obtient un graphe
%         de la derivee.
xp = x + h;
y = feval(fonc,x);
yp = feval(fonc,xp);
Dn = (yp - y)/h;
if nargin>3
```

```
  plot(x,Dn)
  axis(1.1*[min(x), max(x), min(Dn), max(Dn)]);
end
```

`nargin` renvoie le nombre de variables utilisées dans l'appel à la fonction.

Nombre variable d'arguments (*)

La fonction dont on veut calculer la dérivée peut dépendre de paramètres, comme la fonction $F_{(a,b)}$ ci-dessus. On utilisera dans ce cas la fonction `varargin`, comme le montre l'exemple ci-dessous :

```
function Dn = Dernumter(fonc,x,h,dessin,varargin);
% Cette fonction calcule la derivee "numerique" de la fonction
% fonc au(x) point(s) x pour le pas h.
% L'appel se fait selon :
%            Dn = Dernumter(fonc, x, h, dessin, p1, p2,...)
% Les variables en entree sont :
%         - fonc est le nom de la fonction dont on veut calculer
%         la derivee,
%         - x est une valeur reelle ou un vecteur contenant
%         des valeurs reelles du domaine de definition de fonc,
%         - h est le pas du calcul,
%         - dessin est une variable optionnelle ; si on
%         passe une valeur , on obtient un graphe
%         de la derivee.
%         - p1, p2,...  sont d'eventuels parametres de la
%         fonction fonc. On peut passer des parametres sans
%         demander de dessin en utilisant la matrice vide [].
if nargin <= 3; dessin = []; end;
xp = x + h;
y = feval(fonc,x,varargin{:});
yp = feval(fonc,xp,varargin{:});
Dn = (yp - y)/h;
if ~isempty(dessin)
  plot(x,Dn)
  axis(1.1*[min(x), max(x), min(Dn), max(Dn)]);
end
```

`varargin` désigne une liste de longueur quelconque de variables qui peuvent être de tout format ; elle est gérée comme une **cellule de tableaux**. Les cellules de tableaux sont un des moyens proposés par MATLAB pour gérer une collection de variables de types et formats différents. Les lignes suivantes définissent une cellule de tableaux et font s'afficher quelques-uns de ses éléments :

```
>> C ={1:3, rand(2),'coucou'};
>> C{1}
```

```
ans =
     1     2     3
>> c = C{3}
c =
coucou
```

Les éléments d'une cellule sont ordonnés comme ceux d'une matrice, par ligne et par colonne ; on y fait référence par l'utilisation d'accolades au lieu de parenthèses.

Plusieurs arguments en sortie

On peut redéfinir la fonction `F2` de façon qu'elle fournisse aussi la dérivée F' de la fonction F :

$$F'(x) = 0.01\, e^x - 10\, \sin x - 3.$$

Cela peut se faire de la façon suivante :

```
function [y, yprime] = F2bis(x);
% fichier fonction d\'efinissant la fonction F(x) qui sert d'exemple
% au polycopie d'initiation a Matlab, ainsi que sa derivee.
% L'appel se fait selon :
%                       y = F2bis(x);
% et renvoie alors seulement les valeurs de F, ou selon
%                     [y, yprime] = F2bis(x);
y = 0.01*exp(x)+10*cos(x)-3*x;
yprime  = 0.01*exp(x) -10*sin(x) -3;
```

Les deux arguments en sortie sont placés entre crochets. Dans les deux formats d'appel, on calcule $F(x)$ et $F'(x)$. On peut ne faire calculer que $F(x)$ si l'on ne veut pas les valeurs de la dérivée :

```
function [y, yprime] = F2ter(x);
% fichier fonction d\'efinissant la fonction F(x) qui sert d'exemple
% au polycopie d'initiation a Matlab, ainsi que sa derivee.
% L'appel se fait selon :
%                       y = F2ter(x);
%     ou     [y, yprime] = F2ter(x);
% et renvoie alors seulement les valeurs de F, ou selon
%                     [y, yprime] = F2ter(x);
y = 0.01*exp(x)+10*cos(x)-3*x;
if nargout==2
  yprime  = 0.01*exp(x) -10*sin(x)-3;
end
```

1.6.3 Fichiers de sauvegarde

Il est possible de sauvegarder des variables de l'espace de travail en vue d'une utilisation ultérieure. Cela se fait à l'aide de l'instruction `save`. Le format le plus simple est le suivant :

```
save nomfichier X Y Z
```

Les variables `X, Y` et `Z` seront sauvegardées dans un fichier `nomfichier.mat`. On peut les retrouver par l'appel :

```
load nomfichier
```

1.7 Utiliser les aides en ligne

1.7.1 La recherche par mot clef

La commande `lookfor` est une commande de recherche par mot clef. Elle permet de retrouver toutes les fonctions MATLAB dont les commentaires introductifs contient une chaine de caractères donnés. Si, par exemple, je cherche une fonction permettant de tracer un histogramme, je peux entrer l'instruction :

```
lookfor histogram
```

En réponse, j'obtiens la liste suivante :

```
HIST  Histogram.
HISTC Histogram count.
ROSE   Angle histogram plot.
```

On utilise ensuite la commande `help` pour plus de précisions. Cette commande affiche les commentaires de la fonction indiquée. Ces commentaires contiennent l'objet de la fonction ainsi que les différents formats d'appel. Ils contiennent également une rubrique "See Also" qui donne les noms de fonctions dont l'objet est en rapport avec la fonction considérée. Ces commentaires sont affichés sur la fenêtre de travail.

La fonction `helpwin` ouvre une fenêtre qui donne accès à ces commentaires sur une fenêtre séparée. L'accès est arborescent, partant d'un classement par domaine d'application jusqu'à l'aide de chaque fonction. Le lien est fait avec les aides des renvois suggérés par la rubrique "See Also".

1.7.2 La documentation hypertexte

La commande `helpdesk` donne accès à une documentation hypertexte complète supportée par le navigateur internet. Il suffit alors de se laisser guider dans sa recherche.

1.8 Expressions et fonctions MATLAB utilisées

```
axis
clear all
clf
disp
besselj
diag
```

```
eye
eig
Emacs
for
function
global
grid
help
hold off
hold on
if
inline
inv
linspace
mesh
norm
ones
plot, plot3
polyval
print
rand
repmat
reshape
size
surf
title
trill
triu
xlabel
ylabel
while
zeros
```

Chapitre 2

TP1 : Programmation structurée, tracés de courbes (4 séances TP)

Objectif

Le but de ce TP est d'approfondir vos connaissances dans les domaines de la programmation structurée et du tracé de courbes dans MATLAB. Une application illustrant graphiquement la notion de développement de Taylor d'une fonction est proposée. Vous aurez à définir des fichiers `scripts` également appelés `M-Files` et des fonctions. Vous écrirez des boucles, vous aborderez les notions d'argument, de variables locales, globales, muettes.

Avant toute chose vous lirez la documentation relative aux `functions` et fichiers `scripts` (`help function`, `help script`).

2.1 Méthodologie

- Créer le répertoire $TP1$ qui contiendra tous les scripts et toutes les fonctions à programmer. Nommer `main.m` le script principal qui regroupe les lignes de commandes principales de la programmation du TP.
- Pour trouver de l'aide sur une fonction MATLAB ou rechercher le nom d'une fonction, lancer les commandes `help` et `lookfor` (`help nom_de_la_commande` dans la fenêtre de commande MATLAB. Exemple : `help eig`). Regarder à la fin de l'aide, le paragraphe `See also` pour connaître le nom des fonctions similaires.
- Ajouter commentaires et indentation pour améliorer la lisibilité du code pour un usage ultérieur.
- L'utilitaire d'édition *imposé* estt l'éditeur `Emacs`.
- **Il est fortement conseillé de travailler par écrit sur une feuille (algorithme) avant de commencer l'écriture du programme avec le langage** MATLAB.
- **Il est aussi fortement conseillé de tester les lignes de commandes au fur et à mesure de la construction de votre programme**
- Pensez à écrire un programme principal condensé et à faire appel à des fonctions aussi souvent que possible.
- **Un programme principal commence toujours par les instructions :**
  `clear all` : efface toutes les variables en mémoire
  `clf` : efface toutes les figures

2.2 Premiers pas

On s'intéresse aux fonctions de tracés proposées par MATLAB. Pour illustrer ces fonctionnalités, on considère une famille de fonctions définie par :

$$f_\alpha(x) = 1 + \Big(\alpha\, x\, \cos(\alpha\, x)\Big)^2 \tag{2.1}$$

Traçons cette fonction pour $x \in [0, 3]$ et pour trois valeurs du paramètre α. Dans cette première partie "Premiers Pas", le travail proposé consiste à utiliser des boucles au sein d'un fichier `script` unique. **Vous travaillerez donc en consignant les commandes** MATLAB **dans un fichier script dénommé** `script0.m`.

1. Construire un vecteur x composé de 50 points équirépartis entre 0 et 3.

2. A l'aide d'une boucle sur les éléments du vecteur x, construire un vecteur y image de x par la fonction f_α pour $\alpha = 1$ (`help for`).
3. Remplacer la boucle `for` utilisée précédemment par l'opérateur `'.*'`. Expliquer la différence entre `'.*'` et `'*'`.
4. Tracer la fonction f_α sur l'intervalle $[0, 3]$ pour des valeurs du paramètre égales à 1, 2 et 3 sur la même figure. Chacune des courbes devra être tracée avec une couleur différente (`help hold on`, `help plot(x1,y1,x2,y2,x3,y3)`).
5. Modifier le programme afin de réaliser une pause entre les tracés, suspendre l'exécution du script jusqu'à ce que l'utilisateur presse la touche "Entrée". Vous indiquerez un message expliquant à l'utilisateur comment reprendre l'exécution du programme (`help display`, `pause`).

2.3 Définition de fonctions

L'écriture répétitive de $y = 1 + (\alpha\, x\, cos(\alpha\, x))^2$ est fastidieuse et source d'erreurs. Elle peut être évitée en écrivant un programme principal qui fait appel à un autre fichier contenant la définition de cette fonction. Une fonction est écrite dans un fichier différent de celui du "programme principal". En langage MATLAB le nom du fichier de la fonction doit être le même que celui de la fonction. Une fois créée, cette fonction peut être appelée à partir de n'importe quel autre programme principal ou fonction. La notion de fonction est donc associée à l'idée de *bibiothèque de fonctions*. Votre intérêt est de créer des fonctions les plus générales possibles et d'y associer un mode d'emploi en plaçant des commentaires sur la deuxième ligne du fichier fonction.
Exemple : la fonction `toto(x,y)` sera définie dans le fichier `toto.m`. Les premières lignes de ce fichier doivent être :

```
function resultat = toto(x,y)
% fonction sommant ses arguments
resultat = x+y;
```

Dans les autres programmes, cette fonction est appelée en précisant la valeur des arguments. Les arguments x et y sont des variables dites *muettes* : elles servent à réserver une zone mémoire qui reçoit les valeurs utilisées lors de l'appel de la fonction par le programme principal. L'exemple ci-dessous devrait vous permettre de mieux comprendre ce point.

```
% programme principal appelant la fonction toto
...
% avec des valeurs
Somme1 = toto(4,3)
...

% ou des noms de variables après affectation
a = 1; b = 2;
Somme2 = toto(a,b)
...
```

Ici, x, premier argument de la fonction `toto(x,y)`, reçoit la valeur de la variable a au moment où la fonction est appelée, soit 1.

1. Définir la fonction MATLAB `fa2arg.m` qui prend comme arguments d'entrée x et α, et qui retourne la valeur de $f_\alpha(x)$ (`help function`). Les premières lignes du fichier `fa2arg.m` devront être les suivantes :

   ```
   function R = fa2arg(x,Alpha)
   % fonction calculant 1+(Alpha*x*cos(Alpha*x))^2
   % Appel
   % y = fa2arg(x1,Alpha1)
   ...
   ```

   Tester votre fonction.

Une autre possibilité est de créer une fonction avec un seul argument x et de définir le paramètre α comme variable *globale*. Une variable globale permet de communiquer une valeur entre le programme principal et les fonctions sans la passer comme argument de fonction.
La variable globale `Alpha` est visible depuis tout programme ou fonction qui contiendra l'instruction `global Alpha` (`help global`). Cette instruction doit être présente dans les *deux* programmes entre lesquels on souhaite échanger la valeur de la variable.

2. Écrire une `fonction` matlab `fa1arg.m` qui prend comme *unique* argument d'entrée x ; le paramètre (noté α) étant traité comme une variable `globale`.

   ```
   function Res = fa1arg(x)
   % fonction retournant 1+(Alpha*x*cos(Alpha*x))^2
   % Le paramètre Alpha est une VARIABLE GLOBALE
   % Appel
   % y = fa1arg(x1)
   global Alpha
   ...
   ```

3. Évaluer votre fonction avec un argument scalaire.
4. *Les fonctions* MATLAB *utilisent couramment des arguments non scalaires.* Tester votre fonction en passant un argument x de type vecteur. Expliquer le message d'erreur résultant. Procéder aux éventuelles corrections pour que cette évaluation soit possible.

2.4 Représentation de fonctions

Vous travaillerez pour cette partie en consignant les commandes dans le fichier script `script1.m`.

1. Représenter la fonction `fa` sur l'intervalle $[1, 10]$ pour $\alpha = 1$ (on pourra utiliser 50 points d'évaluation) (`help plot`).
2. Ajouter sur la même figure la représentation de la fonction pour $\alpha = 3$ (sur l'intervalle $[1, 10]$) en utilisant une ligne de style et de couleur différents de celle du premier tracé

(`help hold on`, `doc linespec` si vous utilisez la version complète de MATLAB sinon les styles de lignes sont présentés dans l'aide de la fonction `plot`).

3. Effacer la fenêtre graphique (`help clf`). Représenter sur cette figure la fonction pour $\alpha = 2$ et $\alpha = 3$ sur l'intervalle $[0, \pi]$. Vous utiliserez une *échelle logarithmique* pour l'axe des ordonnées (`help semilogy`).
4. Ajouter une légende à la figure *en vous assurant qu'elle ne masque pas les tracés* (`help legend`). Ajouter un titre (`help title`) et préciser les quantités représentées par chacun des axes. Le texte écrit devra avoir une taille de police de 16 (`help FontSize`).

   ```
   title('titre du trace','Fontsize',16);
   ```

5. Fixer la taille de l'axe en x à l'intervalle d'étude $[0, \pi]$. Vous laisserez l'axe y inchangé (`help axis`).
6. Nous allons maintenant représenter la fonction pour quatre valeurs différentes du paramètre α ($\pi/2$, 2π, 3π, 6π) *sur une nouvelle et unique fenêtre graphique*. Pour cela, la figure sera subdivisée en quatre. Vous préciserez pour chaque tracé la valeur du paramètre considéré (les symboles π et α devront apparaître dans le titre) (`help figure`, `help subplot`, `help plot`, `help title`, `help texlabel`).

2.5 Développement de Taylor d'une fonction de plusieurs variables

Vous travaillerez pour cette partie en consignant les commandes dans le fichier script `Taylor.m`. Commencer par écrire trois fichiers "fonction" que vous appelerez ensuite depuis ce programme principal.

Mettons en œuvre le developpement de Taylor (ou Taylor-Young) d'une fonction pour construire le plan tangent à une surface. Considérons une fonction f définie de $\mathbb{R}^2$ dans $\mathbb{R}$. Le graphe de cette fonction définit une surface S dont chaque point M a pour coordonnées $M = (x, y, z)$ avec $z = f(x, y)$).

$$S = \{(x, y, f(x, y)) \,|\, (x, y) \in \mathbb{R}^2$$

On recherche l'équation du plan tangeant à la surface S en un point M_0 de S. Le développement de Taylor de la fonction f au voisinage du point M_0 de coordonnées $(x_0, y_0, z_0) = (H_0, z_0)$ avec $z_0 = f(x_0, y_0)$ et $H_0 = (x_0, y_0)$ est donné par

$$f(H_0 + h) = f(x_0, y_0) + h^T.\nabla f(x_0, y_0) + o(|h|) \tag{2.2}$$

où $\nabla f(x_0, y_0)$ est appelé vecteur gradient de f au point $H_0 = (x_0, y_0)$ et est donné par

$$\nabla f(x_0, y_0) = \left(\frac{\partial f}{\partial x}(x_0, y_0), \frac{\partial f}{\partial y}(x_0, y_0)\right)^T \tag{2.3}$$

et $h = (h_1, h_2)^T$ est un vecteur accroissement. On rappelle que la notation $o(|h|) = |h|\varepsilon(h)$ désigne un reste petit devant $|h|$ et qui peut être négliger en première approximation.

Ainsi on a en développant la formule (2.2)

$$f(x_0+h_1,y_0+h_2)=f(x_0,y_0)+h_1.\frac{\partial f}{\partial x}(x_0,y_0)+h_2.\frac{\partial f}{\partial y}(x_0,y_0)+o(|h|) \tag{2.4}$$

On en déduit que le plan tangeant à S en M_0 a pour équation

$$z=f(x_0,y_0)+x\frac{\partial f}{\partial x}(x_0,y_0)+y\frac{\partial f}{\partial y}(x_0,y_0)$$

L'objectif de cette partie est de vérifier que le vecteur gradient de f évalué en (x_0,y_0) permet de construire une base du plan tangent à la surface au point (x_0,y_0). Cette base est donnée, par exemple, par les vecteurs $\vec{u}$ et $\vec{v}$ suivants

$$\vec{u}=\frac{\partial\overrightarrow{OM}}{\partial x}=\left(1,0,\frac{\partial f}{\partial x}(x_0,y_0)\right)\quad\text{et}\quad\vec{v}=\frac{\partial\overrightarrow{OM}}{\partial y}=\left(0,1,\frac{\partial f}{\partial y}(x_0,y_0)\right) \tag{2.5}$$

On se propose d'étudier la fonction suivante

$$f(x,y)=\frac{x^3}{2}+\frac{y^2}{2} \tag{2.6}$$

Construction des fichiers fonction f et ∇f

1. Écrire une `fonction` matlab `f3d` qui prend comme *unique* argument le vecteur H et qui retourne la valeur de la fonction f au point de coordonnées $H=(x,y)$

   ```
   function Res = f3d(H)
   % fonction retournant la valeur z=f(x,y)
   Res=
   ```

2. Écrire une `fonction` matlab `gradf` qui prend comme argument le vecteur H et qui retourne le vecteur gradient de la fonction f au point de coordonnées $H=(x,y)$

   ```
   function Res = gradf(H)
   % fonction retournant le vecteur [df/dx; df/dy]
   Res=
   ```

3. Écrire une `fonction` matlab `plantangent` qui prend comme arguments les deux vecteurs h et H et qui retourne l'approximation de $f(H+h)$ calculée par la formule de Taylor (2.2).

   ```
   function Res = plantangent(X,h)
   % fonction retournant la valeur de f en X+h
   Res=
   ```

Tracé de la surface en 3D

Vous travaillerez ici dans le fichier script `Taylor.m`. Construire sur papier l'organigramme de ce programme qui devra afficher sur un même graphique, la surface, un point de cette surface, les vecteurs tangents à la surface et le plan tangent en ce point. Vous ferez appel aux 3 fonctions déjà créées et vous vous aiderez des 4 premières questions qui suivent.

1. Construire deux vecteurs x et y *de même taille* pour $x \in [-2, 2]$ et $y \in [-3, 3]$. A l'aide de la fonction `f3d`, construire la matrice $Z(i, j) = f(x(i), y(j))$. Tracez la surface $z = f(x, y)$ (`help surf(y,x,Z)`.
   Note : attention à l'ordre des vecteurs, le premier vecteur correspond aux colonnes de Z
2. Placer sur ce graphique un point de cordonnées $(x_0, y_0, f((x_0, y_0))$ au choix (`help plot3`).
3. Tracer les deux vecteurs formant la base du plan tangent en ce point (`help plot3, markerfacecolor, markersize`).
4. Calculer les coordonnées des points appartenant au plan tangent en $(x_0, y_0, f((x_0, y_0))$ dans un voisinage de plus ou moins 0,5 dans les directions x et y. Tracer ce plan sur le même graphique que la surface précédente.
5. Tester votre programme pour différents points (x_0, y_0).
6. Améliorer la convivialité en demandant à l'utilisateur de saisir les coordonnées du point (x_0, y_0) (`help input`).

2.6 Annotation d'un tracé, fonctionnalités avancées

Nous allons utiliser les *identifiants* (*handle* pour MATLAB) des objets graphiques afin de modifier leurs propriétés. Vous *travaillerez dans un nouveau fichier script* `script3.m`.

1. (*) Fermer toutes les fenêtres graphiques. Exécuter le fichier `script1`. Sélectionner la fenêtre 1 et modifier la propriété `FontSize` des axes de la figure afin d'obtenir des polices de taille 16 (`help gca`, `set`).
2. (*) Sélectionner la fenêtre 3. Créer une nouvelle légende identique à la précédente, mais en stoquant l'identifiant de cet objet graphique dans la variable `Legend_Handle`. Changer la taille des polices utilisées pour la légende (`help set`).
   Remarque : La création d'une nouvelle légende détruit l'ancienne (l'objet légende est unique pour chaque figure). Il est possible de récupérer l'identifiant de la légende grâce à la commande `>> findobj(gcf,'Tag','legend')`
3. Annoter une figure. Nous allons ajouter sur la figure un texte indiquant la valeur prise par la fonction en un point de l'intervalle.
   (a) Exécuter les instructions MATLAB suivantes :

   ```
   figure; Z='\alpha'; Y='\approx' ; X = '\rightarrow';
   Mon_Texte = sprintf('%s','f_',Z,Y,X);
   Text_Handle = text(0.5,0.5,Mon_Texte);
   ```

   Que contiennent les variables `Mon_Texte` et `Text_Handle` ? (`help sprintf`, `text`).
   (b) Quelle est la signification du résultat de l'instruction `>>get(Text_Handle,'FontSize')` (`help get`).
   (c) (*) Changer la taille du texte identifié par `Text_Handle` (`help set`).
   (d) Effacer ce texte (`lookfor delete`).

(e) (*) Construire une chaîne de caractères de la forme "$f_Z(X) \approx Y \rightarrow$" où X, Y, Z sont des chaînes de caractères définies dans des variables (Exemple : `>> Z = '2'; X='\pi'`).
Indications : `sprintf`, les symboles $\approx$ et $\rightarrow$ sont obtenus grâce à respectivement `\approx` et `\rightarrow`).

(f) (**) Procéder aux modifications nécessaires pour prendre en compte une variable `Z` et `Y` numériques (`help num2str`).

2.7 Expressions et fonctions MATLAB utilisées

`clear all`
`clf`
`help`
`script`, `function`
boucle `for`
`hold on`
`display`
`pause`, `input`
`global`
`plot`, `plot3`, `surf`
`subplot`, `markerfacecolor`, `markersize`
`axis`, `legend`, `title`, `label`
`FontSize`, `semilogy`
`texlabel`, `figure`
`gca`, `set`, `get`
`sprintf`, `text`

Chapitre 3

TP2 : Intégration numérique - Sensibilité à la discrétisation (2 séances TP)

Objectif

L'objet de ce TP est le calcul numérique sous MATLAB de l'intégrale sur un segment $[a, b]$ d'une fonction d'une variable par la méthode du point milieu et la méthode des trapèzes. Soit $(a,\, b) \in \mathbb{R}^2$ tel que $-\infty < a < b < +\infty$. Soit $f : [a,\, b] \to \mathbb{R},\ x \mapsto f(x)$ une fonction continue donc intégrable sur $[a,\, b]$.

On se propose de calculer numériquement une valeur approchée de l'intégrale définie

$$I(f;\, a,\, b) = \int_a^b f(x)\, dx \tag{3.1}$$

Comme l'intégrale représente l'aire algébrique du domaine située entre la courbe $y = f(x)$ et l'axe des abscisses et entre les deux droites $x = a$ et $x = b$, on va approcher ce domaine par un découpage en carrés

3.1 Méthodologie

- Créer le répertoire $TP2$ qui contiendra tous les scripts et toutes les fonctions à programmer. Nommer `main.m` le script principal qui regroupe les lignes de commandes principales de la programmation du TP.
- Pour trouver de l'aide sur une fonction MATLAB ou rechercher le nom d'une fonction, lancer les commandes `help` et `lookfor` (`help nom_de_la_commande` dans la fenêtre de commande MATLAB. Exemple : `help eig`). Regarder à la fin de l'aide, le paragraphe `See also` pour connaître le nom des fonctions similaires.
- Ajouter commentaires et indentation pour améliorer la lisibilité du code pour un usage ultérieur.
- L'utilitaire d'édition *imposé* estt l'éditeur `Emacs`.
- **Il est fortement conseillé de travailler par écrit sur une feuille (algorithme) avant de commencer l'écriture du programme avec le langage** MATLAB.
- **Il est aussi fortement conseillé de tester les lignes de commandes au fur et à mesure de la construction de votre programme**
- Pensez à écrire un programme principal condensé et à faire appel à des fonctions aussi souvent que possible.
- **Un programme principal commence toujours par les instructions :**
  `clear all` : efface toutes les variables en mémoire
  `clf` : efface toutes les figures

3.2 Méthode du point milieu

3.2.1 Eléments théoriques

Soit $a = x_0 < x_1 < \cdots < x_m = b$ un découpage de l'intervalle d'étude en m segments égaux. On note $h = x_{i+1} - x_i = \frac{b-a}{m}$ la longueur de chaque petit segment. La méthode du point milieu consiste à approcher la fonction f par une constante sur chaque intervalle de

la forme $[x_i, x_{i+1}[$.

$$\forall i = 0 \dots m-1\,, \quad \forall x \in [x_i, x_{i+1}[\,, \quad f(x) \approx f(c_i), \text{ avec } c_i = \frac{x_{i-1} + x_i}{2}$$

où c_i est le centre de l'intervalle $[x_i, x_{i+1}[$. La valeur approchée de l'intégrale est alors

$$I_m(f;\, a,\, b) = \sum_{i=1}^{m} h * f(c_i) = h \sum_{i=1}^{m} f(c_i)\ . \tag{3.2}$$

FIG. 3.1 – Approximation de l'aire I(f ; a, b) par la méthode du point milieu

On souhaite évaluer l'erreur commise lors de cette approximation :

$$E_m(f; a; b) = I(f; a; b) - I_m(f; a; b)$$

On suppose que $f \in C^2([a,\, b])$ et on écrit le développement de Taylor de f à l'ordre 2 autour du point c_i. Il existe $\xi_i \in]x_{i-1},\, x_i[$ tel que

$$f(x) = f(c_i) + f'(c_i)(x - c_i) + \frac{1}{2} f''(\xi_i)(x - c_i)^2$$

le point ξ_i dépendant du point x. En intégrant entre x_{i-1} et x_i, il vient par la formule de la moyenne

$$I(f;\, x_{i-1},\, x_i) = I_m(f;\, x_{i-1},\, x_i) + 0 + \frac{1}{2} f''(\eta_i) \left[\frac{(x - c_i)^3}{3} \right]_{x_{i-1}}^{x_i}\ ,$$

où $\eta_i \in]x_{i-1},\, x_i[$. Ainsi, si on note M un majorant de f" sur $[a, b]$, on a

$$E_m(f;\, a,\, b) \leq M \frac{b-a}{24} h^2\ . \tag{3.3}$$

3.2.2 Algorithme et programmation

Lisez l'algorithme de calcul de l'intégrale par la méthode du point milieu $I_m(f; a, b)$:

```
% Initialisation
quadrature = 0
% Calcul du pas h
Evaluer  h = (b - a) / m
% Début de la boucle servant à parcourir l'intervalle [a,b]
Pour i de 1 à m
Evaluer c(i) = a + h * (2*i - 1) / 2
Evaluer quadrature = quadrature + f(c(i))
% Fin de la boucle
% Attention à l'implémentation de f(c(i)) ...
% Calcul de l'intégrale
Evaluer quadrature = h * quadrature
```

3.2.3 Implémentation

L'implémentation du calcul de l'intégrale d'une fonction f, dont le nom est stocké dans la variable MATLAB `fun`, implique l'écriture de trois fichiers :

- un fichier script du programme principal qui interroge l'utilisateur sur le nom `fun` de la fonction à intégrer, la valeur des bornes `a` et `b`, le nombre `m` de sous-intervalles de discrétisation, la méthode d'intégration et qui réalise l'appel des fonctions.
- un fichier de type `function` qui calcule l'image par la fonction f de x.
- un fichier de type `function` qui décrit la méthode de calcul utilisée.

Pour ce faire :

1. Implémenter sous MATLAB l'algorithme ci-dessus dans le fichier fonction `fquadpm1.m`. La première ligne de ce fichier est donnée par :

   ```
   function quadrature = fquadpm1(fun, a, b, m)
   ```

   L'argument `fun` désigne soit le nom de la fonction générique à intégrer stocké dans une chaîne de caractères soit un pointeur sur cette fonction (`help function`, `help feval`).

2. Comparer cette implémentation (faire un commentaire) avec celle donnée ci-dessous que l'on vous demande de reproduire dans le fichier `fquadpm2.m` (`help ops`, `help colon`).

   ```
   function quadrature = fquadpm2(fun, a, b, m)
   h = (b - a) ./ m;
   x = a + h./2:h:b;
   y = feval(fun, x);
   quadrature = h .* sum(y);
   ```

3. On étudie le cas $(a, b) = (0, 1)$ et $f : [0, 1] \to \mathbb{R},\ x \mapsto f(x) = e^{-x^2}$. Créer un fichier `function` nommé `fexpmx2.m` pour définir la fonction f de la variable x.

Indications supplémentaires :

La commande `input` permet d'interagir avec l'utilisateur pour entrer des données dans la fenêtre principale de MATLAB au cours de l'exécution d'un script ou d'une fonction. La commande `switch...case...end` peut être utile pour orienter le calcul de quadrature en fonction de la méthode.

3.2.4 Application numérique

Faire en sorte que le programme principal affiche clairement tous les résultats demandés ci-après.

1. Calculer $I_m(e^{-x^2};\, 0,\, 1)$ pour $m = 10$ et $m = 100$, à l'aide de la fonction `fquadpm1.m`.
2. Idem avec `fquadpm2.m`. Vérifier que les résultats obtenus coïncident.
3. On veut estimer une majoration de l'erreur de quadrature pour $m = 10$ et $m = 100$ en valeur absolue. A cette fin, calculer à la main $f''(x)$, tracer avec MATLAB, $x \mapsto f''(x)$ et en déduire visuellement une valeur de $M_2(f) = max_{[a,\,b]}|f(x)|$ par excès. En déduire une majoration numérique de $|E_m(e^{-x^2};\, 0,\, 1)|$ pour $m = 10$ et $m = 100$.

3.3 Méthode des trapèzes

3.3.1 Eléments théoriques

La méthode des trapèzes consiste à approcher la fonction non plus par une constante mais par une fonction affine par morceau sur chaque intervalle de la forme $[x_i, x_{i+1}[$:

$$\forall i = 0 \dots m-1\,, \quad \forall x \in [x_i, x_{i+1}[\,, \quad f(x) \approx f(x_i)\left(\frac{x_{i+1}-x}{h}\right) + f(x_{i+1})\left(\frac{x-x_i}{h}\right)\ .$$

La valeur approchée de l'intégrale est alors

$$I'_m(f;\, a,\, b) = \frac{h}{2}\sum_{i=1}^{m}(f(x_{i-1}) + f(x_i))\ . \tag{3.4}$$

On démontre que l'erreur de quadrature s'écrit, lorsque $f \in C^2([a,\, b])$,

$$E'_m(f;\, a,\, b) = I(f;a;b) - I'_m(f;a;b) = -\frac{b-a}{12}h^2 f''(\xi)\ , \tag{3.5}$$

pour un certain $\xi \in [a,\, b]$.

Remarque : La méthode du point milieu - à ne pas confondre avec la méthode des rectangles à droite ou à gauche - de part le coefficient $\frac{1}{24}$, reste plus précise que la méthode des trapèzes dont l'erreur est en $\frac{1}{12}$, toutes les autres grandeurs égales par ailleurs, au signe près.

En fait, la méthode du point milieu est, comme la méthode des trapèzes, une méthode d'ordre 1, i.e., montre une erreur en h^2. (L'erreur d'une méthode d'ordre p est en h^{p+1}). Ceci est dû au fait que l'aire du rectangle $h.f(\frac{x_i+x_{i+1}}{2})$ coïncide avec l'aire du trapèze coloré $h.\frac{1}{2}.(f(x_i) + f(x_{i+1}))$ pour une application $f : x \mapsto Ax + B$; la formule approchée devient donc exacte si f est une fonction affine.

FIG. 3.2 – Approximation de l'aire I(f ; a, b) par la méthode des trapèzes

FIG. 3.3 – La méthode du point milieu intègre exactement une droite affine

3.3.2 Algorithme

1. En s'inspirant de l'exemple de la méthode du point milieu, écrire un algorithme pour le calcul de l'intégrale par la méthode des trapèzes.

3.3.3 Implémentation

2. Implémenter cet algorithme sous MATLAB dans un fichier fonction `fquadtrap.m` **sans** utiliser de boucle `for` ou `while` comme dans l'exemple de la fonction `fquadpm2.m`. La première ligne du fichier `fquadtrap.m` est donnée par :
   `function quadrature = fquadtrap(fun, a, b, m)`

3.3.4 Application numérique

1. Compléter le programme principal afin de calculer également la valeur de l'intégrale $I_{1,m}(e^{-x^2}; 0, 1)$ pour $m = 10$ et $m = 100$ à l'aide de la fonction `fquadtrap.m`.
2. Evaluer une majoration de $|E'_m(e^{-x^2}; 0, 1)|$ pour $m = 10$ et $m = 100$.
3. Calculer une valeur approchée de l'intégrale de $f : x \mapsto exp(-x^2)$ à l'aide de la fonction `quad.m` de MATLAB.
4. Idem pour $f : x \mapsto exp(-x^3)$ à l'aide d'une implémentation `inline` (`help inline`).
5. Dresser un tableau récapitulatif de tous les résultats.
6. Commenter.

3.4 Expressions et fonctions MATLAB utilisées

`help`
`lookfor`
`who, whos`
`function`
`sum`
`feval`
`quad`
`plot`
`for...end`
`while...end`
`switch...case...end`
`disp`
`input`
`inline`
Opérateur `:` (`help colon`)
Opérateur `@` (`help function_handle`, `help punct`, et plus généralement `help ops`).

Chapitre 4

TP3 : Recherche des zéros d'une fonction réelle à variable réelle (2 séances TP)

4.1 Objectif

L'objet de ce TP est de résoudre, grâce à la méthode de dichotomie, l'équation

$$f(x) = 0 \tag{4.1}$$

où f est une application continue de $\mathbb{R}$ dans $\mathbb{R}$ et x l'inconnue recherchée.

4.2 Méthodologie

- Créer le répertoire $TP3$ qui contiendra tous les scripts et toutes les fonctions à programmer. Nommer `main.m` le script principal qui regroupe les lignes de commandes principales de la programmation du TP.
- Pour trouver de l'aide sur une fonction MATLAB ou rechercher le nom d'une fonction, lancer les commandes `help` et `lookfor` (`help nom_de_la_commande` dans la fenêtre de commande MATLAB. Exemple : `help eig`). Regarder à la fin de l'aide, le paragraphe `See also` pour connaître le nom des fonctions similaires.
- Ajouter commentaires et indentation pour améliorer la lisibilité du code pour un usage ultérieur.
- L'utilitaire d'édition *imposé* estt l'éditeur `Emacs`.
- **Il est fortement conseillé de travailler par écrit sur une feuille (algorithme) avant de commencer l'écriture du programme avec le langage** MATLAB.
- **Il est aussi fortement conseillé de tester les lignes de commandes au fur et à mesure de la construction de votre programme**
- Pensez à écrire un programme principal condensé et à faire appel à des fonctions aussi souvent que possible.
- **Un programme principal commence toujours par les instructions :**
  `clear all` : efface toutes les variables en mémoire
  `clf` : efface toutes les figures

4.3 Méthode de dichotomie (sans test de bornes)

4.3.1 Eléments théoriques

Soit f une fonction réelle d'une variable réelle, continue. On appelle zéros de f les solutions, si elles existent, de l'équation (4.1). La méthode de dichotomie se base sur la proposition suivante :

Proposition 1. *Soit* $[a, b]$ *un segment de* $\mathbb{R}$. *Soit* f *une fonction continue de* $[a, b]$ *dans* $\mathbb{R}$ *telle que* $f(a)f(b) < 0$. *Alors il existe un zéro* $\lambda \in]a, b[$ *de* f *tel que* $f(\lambda) = 0$.

Le but de la méthode est donc de choisir convenablement deux réels a et b de telle sorte que l'intervalle $[a, b]$ contienne un zéro et un seul de la fonction f. Il faudra notamment que $f(a)$ et $f(b)$ soient de signes opposés. Décrivons maintenant l'algorithme de dichotomie.

L'idée est, d'une manière itérative, de créer une suite d'intervalles emboîtés qui contiennent

un zéro de f et dont la longueur est divisée par deux à chaque itération. On démarre avec le premier intervalle $I_0 = [a_0, b_0]$ avec $a_0 = a$ et $b_0 = b$ puis on définit la suite de sous-intervalles $I_k = [a_k, b_k]$ tels que $I_k \subset I_{k-1}$, $\forall k \in \mathbb{N}^*$ et $f(a_k)f(b_k) < 0$,

$$I_n \subset ... \subset I_k \subset I_{k-1} ... \subset I_0 \ .$$

A partir des données de l'itération k, le point central de l'intervalle I_k est

$$x_k = \frac{1}{2}(a_k + b_k) \ .$$

Puis, on effectue le choix suivant :

$\diamond$ soit $f(a_k)f(x_k) < 0 : a_{k+1} = a_k$ et $b_{k+1} = x_k$,
$\diamond$ soit $f(x_k)f(b_k) < 0 : a_{k+1} = x_k$ et $b_{k+1} = b_k$.

Enfin, on pose

$$x_{k+1} = \frac{1}{2}(a_{k+1} + b_{k+1}) \ .$$

Les cas théoriques où $f(a_k)f(x_k) = 0$ ou $f(x_k)f(b_k) = 0$ ne sont pas traités ici.

La méthode de dichotomie se termine à l'itération n définie par $|x_n - \lambda| \leq |I_n| \leq \varepsilon$ pour une précision ε fixée à l'avance.

On note $|I_n|$ la longueur du segment I_n. Alors $|I_0| = b - a$, $|I_1| = (b-a)/2$. On montre ainsi par récurrence que $|I_n| = (b-a)/2^n$.

Considérons maintenant l'erreur absolue $e_k = |x_k - \lambda|$ à l'itération k entre la valeur exacte du zéro $\lambda \in]a, b[$ de la fonction f et la valeur approchée x_k de ce même zéro calculée par l'algorithme à l'étape k.

Puisque, $\forall x \in I_k$, $|x - \lambda| \leq |I_k|$, il vient $|e_k| \leq (b-a)/2^k$; d'où, $\lim_{k \to +\infty} |e_k| = 0$. L'erreur absolue converge donc *globalement*.

4.3.2 Illustration théorique

Afin d'illustrer cette méthode par un exemple simple, considérons la recherche de la racine cubique de 2. Soit $f : x \mapsto x^3 - 2$. Cherchons la racine λ de l'équation $f(x) = 0$.

Avec MATLAB, on trace facilement le graphique de la figure 4.1 pour se donner une idée de la localisation de $\lambda = \sqrt[3]{2}$.

Dans le cas présent, il suffit de se dire $\sqrt[3]{1} < \sqrt[3]{2}$ et $\sqrt[3]{2} < 2$ pour considérer la première étape de l'algorithme :

$\diamond$ `Itération 0` : $a_0 = a = 1$, $b_0 = b = 2$, $x_0 = \frac{a_0+b_0}{2} = \frac{3}{2}$, $f(a_0) = -1$, $f(b_0) = 6$, $f(x_0) = \frac{11}{8}$

Puisque $f(a_0).f(x_0) < 0$, on choisit :

$\diamond$ `Itération 1` : $a_1 = a_0 = 1$, $b_1 = x_0 = \frac{3}{2}$, $x_1 = \frac{a_1+b_1}{2} = \frac{5}{4}$, $f(a_1) = -1$, $f(b_1) = \frac{11}{8}$, $f(x_1) = -\frac{3}{64}$

Puisque $f(x_1).f(b_1) < 0$, on choisit :

FIG. 4.1 – Méthode de dichotomie

◇ `Itération 2` : $a_2 = x_1 = \frac{5}{4}$, $b_2 = b_1 = \frac{3}{2}$, $x_2 = \frac{a_2+b_2}{2} = \frac{11}{8}$, $f(a_2) = -\frac{3}{64}$, $f(b_2) = \frac{11}{8}$, $f(x_2) = \frac{307}{512}$

Puisque $f(a_2).f(x_2) < 0$, on choisit :

◇ `Itération 3` : $a_3 = a_2 = \frac{5}{4}$, $b_3 = x_2 = \frac{11}{8}$, $x_3 = \frac{a_3+b_3}{2} = \frac{21}{16}$, $f(a_3) = -\frac{3}{64}$, $f(b_3) = \frac{307}{512}$, $f(x_3) = \frac{1069}{4096}$

Puisque $f(a_3).f(x_3) < 0$, on choisit :

◇ `Itération 4` : $a_4 = a_3 = \frac{5}{4}$, $b_4 = x_3 = \frac{307}{512}$, $x_4 = \frac{a_4+b_4}{2} = \frac{41}{32}$, $f(a_3) = -\frac{3}{64}$, $f(b_4) = \frac{1069}{4096}$, $f(x_4) = \frac{3385}{32768}$

et ainsi de suite...

Avec $\lambda = x_4 = \frac{41}{32}$, on obtient l'approximation $\sqrt[3]{2} \approx 1.2813$ à comparer avec la valeur donnée par MATLAB :

`2.^(1./3)= 1.2599` avec 4 décimales (mode `format` par défaut).

L'erreur maximale est donnée par la moitié de la longueur de l'intervalle final $|I_4| = |b_4 - a_4| = \frac{333}{512}$, soit $\frac{333}{1024} \approx 0.3252$, précision médiocre que l'on peut améliorer en poussant plus loin le nombre d'itérations.

Il se trouve que cette formulation théorique en particulier (nous n'avons pas fait de calcul dans MATLAB sauf éventuellement à évaluer les numérateurs et dénominateurs des fractions), fonctionne très bien une fois programmée dans MATLAB. Mais, ce n'est pas toujours le cas comme vu ultérieurement.

4.3.3 Algorithme

Rédiger l'algorithme de cette méthode en tenant compte d'un nombre maximal d'itérations possibles afin de sortir d'un calcul divergent.

4.3.4 Implémentation

Programmer cette méthode sous MATLAB. Le calcul de la racine se fera dans un fichier fonction (`help function`) nommé `fdicho.m`. La fonction devra retourner la valeur de la racine ainsi que le nombre d'itérations nécessaires. La première ligne sera donc

`function [lambda, niter] = fdicho(fun, a, b, precision, nitermax)`

où

`fun` : le nom de la fonction générique,
`a` et `b` : les bornes de l'intervalle d'études,
`precision` : ε,
`nitermax` : le nombre maximal d'itérations possibles,
`lambda` : la racine recherchée,
`niter` : le nombre d'itérations nécessaires.

Indications : Utiliser la fonction `feval` (`help feval`).

L'appel à cette fonction est effectué depuis le programme principal.

4.3.5 Application numérique

Faire en sorte que le programme principal affiche clairement tous les résultats demandés ci-après.

On considère la fonction associée au polynôme de Legendre de degré 5,

$$\begin{array}{rl} L_5 : & [-1,\,1] \to \mathbb{R}\ , \\ & x \mapsto L_5(x) = \frac{x}{8}(63x^4 - 70x^2 + 15)\ . \end{array} \tag{4.2}$$

4.3.5.a) Implémenter cette fonction dans MATLAB (`help function`), dans un fichier `fl5.m` qui reçoit comme argument l'abscisse x et qui retourne la valeur $L_5(x)$.

4.3.5.b) Tracer à l'aide de MATLAB (`help plot`) la fonction L_5 et donner dans votre compte-rendu une valeur approchée a 10^{-1} près (calculée visuellement) des cinq zéros de l'équation $L_5(x) = 0$ qui comptent la solution triviale $x = 0$. On démontre que ces racines appartiennent à l'intervalle $]-1,\,1[$. On rangera les racines dans l'ordre croissant $(\lambda_i)_{1\leq i\leq 5}$.

4.3.5.c) Utiliser la fonction `fdicho` pour trouver les racines non triviales avec quatre chiffres significatifs. On choisira `nitermax=100`.

Quels autres paramètres choisissez-vous pour tester `fdicho` ? Pourquoi ?

4.3.5.d) Modifier la fonction `fdicho` afin qu'elle retourne chaque abscisse x_k pour chaque itération effectuée. Depuis le programme principal, tracer alors l'historique de la valeur absolue de l'erreur absolue e_k pour la racine $\lambda_5 \simeq 0.9$, avec en abcisse, l'itération k, en ordonnée, $|e_k|$. Ajouter le titre du graphique (`help title`) et la légende (`help legend`).

L'erreur absolue converge-t-elle de façon monotone en valeur absolue ?

4.3.5.e) Utiliser les fonctions MATLAB `fzero` et `roots` pour trouver les zéros de $L_5(x) = 0$.

4.3.6 Contre-exemple

La méthode fonctionne bien pour les problèmes (équations & intervalle $[a, b]$) vus ci-dessus. Cependant, d'autres problèmes nécessitent quelques aménagements.

4.3.6.a) Discuter sur le compte rendu - **d'un point de vue théorique pur** - de l'application possible ou non, des deux premières itérations de la méthode de dichotomie vue ci-dessus à la recherche des zéros de l'équation $\cos x = 0$ pour les conditions suivantes :

(a) $a_0 = 0$ et $b_0 = \pi$,

(b) $a_0 = 0$ et $b_0 = \pi/4$,

(c) $a_0 = \pi/2$ et $b_0 = \pi$.

4.3.6.b) Discuter sur le compte rendu - **en faisant TOUS les calculs mêmes simples en** MATLAB - de l'application possible ou non, des deux premières itérations de la méthode de dichotomie vue ci-dessus à la recherche des zéros de l'équation $\cos x = 0$ pour les conditions suivantes :

(a) $a_0 = 0$ et $b_0 = \pi$,

(b) $a_0 = 0$ et $b_0 = \pi/4$,

(c) $a_0 = \pi$ et $b_0 = 3\pi/2$.

4.3.6.c) Peut-on dans les cas comparables (a) et (b) appliquer la méthode selon les deux points de vue, théorique et pratique ?

4.3.6.d) Conclusion : que faut-il modifier et / ou ajouter dans la méthode de dichotomie visée ci-dessus pour qu'elle devienne applicable à l'équation $\cos(x) = 0$ quelles que soient les bornes de l'intervalle initial.

4.4 Méthode de dichotomie (avec test de bornes)

4.4.1 Eléments théoriques

Il s'agit d'une part de détecter les valeurs qui sont proches du zéro numérique. En effet, la représentation finie des nombres en machine ne permet pas toujours d'obtenir le 0 absolu dans MATLAB mais une valeur approchée très petite. Cette valeur est approchée à `eps/2` $\approx$ `1.1102e-016` près (`help eps`).

Il s'agit également de vérifier que l'algorithme s'arrête dès lors qu'un premier test permet (à tort ou à raison) d'affirmer qu'aucune racine n'existe dans l'intervalle d'étude : les signes de $f(a_0)$ et de $f(b_0)$ étant identiques.

4.4.2 Algorithme

Rédiger l'algorithme modifié de la méthode de dichotomie qui prend en compte les valeurs proches du zéro numérique et les bornes de même signe.

4.4.3 Implémentation

Programmer ce nouvel algorithme dans un fichier fonction nommé `fdicho_test.m`.

4.4.4 Application numérique

Faire en sorte que le programme principal affiche clairement tous les résultats demandés ci-après.

Par l'usage de `fdicho_test.m`, trouver les racines de l'équation $\cos x = 0$ pour les intervalles suivants :

1. $a_0 = 0$ et $b_0 = \pi$,
2. $a_0 = 0$ et $b_0 = \pi/4$,
3. $a_0 = \pi$ et $b_0 = 3\pi/2$.

4.5 Expressions et fonctions MATLAB utilisées

`help`
`lookfor`
`who, whos`
`function`
`feval`
`fzero, roots`
`plot, title, legend, axis equal, axis square`
`for...end, while...end`
Opérateur : (`help colon`)
Opérateur `@` (`help function_handle`, `help punct`, et plus généralement `help ops`).

4.6 Complément théorique sur la précision de calcul

Pour donner une estimation du nombre d'itérations nécessaires pour le calcul de la racine λ avec une précision ε donnée à l'avance, des inégalités $|x_n - \lambda| \leq |I_n| \leq \varepsilon$, il vient

$$n \geq E\left[\frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln 2}\right] + 1 \ . \tag{4.3}$$

Ainsi, augmenter la précision de calcul d'une décimale, c'est-à-dire, passer de ε à $\varepsilon' = \varepsilon/10$, et n devient $n' = n + E[\frac{\ln 10}{\ln 2}] + 1 = n + 4$.

Autrement dit, quatre itérations supplémentaires permettent d'améliorer d'un chiffre significatif la précision de calcul de la racine.

Chapitre 5

TP4 : Diagonalisation de matrices (2 séances TP)

Objectif

L'objectif du TP est de mettre en œuvre la diagonalisation de matrices et en particulier de montrer sur un exemple l'intérêt de calculer les valeurs propres et les vecteurs propres d'une matrice.

5.1 Méthodologie

- Créer le répertoire $TP4$ qui contiendra tous les scripts et toutes les fonctions à programmer. Nommer `main.m` le script principal qui regroupe les lignes de commandes principales de la programmation du TP.
- Pour trouver de l'aide sur une fonction MATLAB ou rechercher le nom d'une fonction, lancer les commandes `help` et `lookfor` (`help nom_de_la_commande` dans la fenêtre de commande MATLAB. Exemple : `help eig`). Regarder à la fin de l'aide, le paragraphe `See also` pour connaître le nom des fonctions similaires.
- Ajouter commentaires et indentation pour améliorer la lisibilité du code pour un usage ultérieur.
- L'utilitaire d'édition *imposé* estt l'éditeur `Emacs`.
- **Il est fortement conseillé de travailler par écrit sur une feuille (algorithme) avant de commencer l'écriture du programme avec le langage** MATLAB.
- **Il est aussi fortement conseillé de tester les lignes de commandes au fur et à mesure de la construction de votre programme**
- Pensez à écrire un programme principal condensé et à faire appel à des fonctions aussi souvent que possible.
- **Un programme principal commence toujours par les instructions :**
  `clear all` : efface toutes les variables en mémoire
  `clf` : efface toutes les figures

5.2 Quelques rappels mathématiques (*)

Soit Σ une matrice carrée (ou un tenseur) réelle de taille 3×3. Le réel σ est une valeur propre (ou valeur principale ou eigenvalue en anglais) de Σ si il existe un vecteur **non nul** $\overrightarrow{n}$ de $\mathbb{R}^3$ appelé vecteur propre (ou vecteur principal ou eigenvectors en anglais) de Σ tel que

$$\Sigma \overrightarrow{n} = \sigma \overrightarrow{n} \tag{5.1}$$

En pratique, on recherche les valeurs propres de Σ parmi les racines du polynôme caractéristique de Σ noté $P_\Sigma(x) = \det(\Sigma - xI)$. Puis, on trouve les vecteurs propres associés en résolvant le système (5.1). Il n'y a pas unicité du vecteur propre car tout multiple d'un vecteur propre est encore vecteur propre associé à la même valeur propre.

Dans le cas où Σ admet trois valeurs propres distinctes σ_I, σ_{II} et σ_{III}, la matrice Σ est diagonalisable, c'est-à-dire, il existe une matrice inversible $P \in GL_3(\mathbb{R})$ telle que

$$\Sigma = PDP^{-1} = P \begin{pmatrix} \sigma_I & 0 & 0 \\ 0 & \sigma_{II} & 0 \\ 0 & 0 & \sigma_{III} \end{pmatrix} P^{-1}$$

P s'appelle la matrice de changement de base. Cela signifie que si un vecteur $\overrightarrow{v}$ a les composantes (x, y, z) dans le repère de départ appelé repère de référence Ref, il aura les composantes (X, Y, Z) dans le repère des vecteurs propres appelé repère principal RP avec

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = P \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

5.3 Position du problème physique

Pour illustrer ce TP, nous avons choisi un exemple de mécanique des milieux continus. Soit Ref un repère de référence de l'espace. Soit Σ un tenseur de contraintes en un point de coordonnées (x, y, z) relativement à Ref et à un instant t, dans un milieu continu déformable. Le tenseur s'écrit :

$$\mathbf{\Sigma}_{Ref} = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ce point du milieu continu est entouré de matière. On peut donc définir une infinité de facettes passant par ce point. On choisit de repérer l'orientation de chaque facette par sa normale extérieure $\overrightarrow{n}_{Ref}$, il s'agit d'un vecteur de norme 1 de $\mathbb{R}^3$:

$$\overrightarrow{n}_{Ref} = \begin{pmatrix} l \\ m \\ n \end{pmatrix}$$

L'intensité du vecteur contrainte $\overrightarrow{\sigma}\,(\overrightarrow{n}_{Ref})$ qui s'exerce sur la facette de normale $\overrightarrow{n}_{Ref}$ est le produit du tenseur de contraintes Σ par le vecteur $\overrightarrow{n}_{Ref}$ définissant la normale :

$$\overrightarrow{\sigma}(\overrightarrow{n}_{Ref}) = \mathbf{\Sigma}_{Ref}.\overrightarrow{n}_{Ref} = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} l \\ m \\ n \end{pmatrix}$$

Si la normale prend toutes les orientations possibles dans l'espace, l'extrémité du vecteur contrainte décrit *une surface que l'on se propose de déterminer.*

5.4 Programmation

1. Tracer les axes du repère cartésien Réf, pour -4<x<4, -4<y<4, -4<z<4 (`help plot3`). Une fois tracés les axes, faire pivoter la figure.
2. On note (r, θ, ϕ) les coordonnées sphériques dans l'espace $\mathbb{R}^3$. Pour faire parcourir au vecteur normal tout l'espace, on propose de faire varier l'angle θ de 0°à 360°tous les 10°(soient ntheta valeurs) et l'angle ϕ de 0°(au pôle nord) à 180°(au pôle sud) tous les 10°aussi (soient nphi valeurs). Pour chaque valeur de θ et de ϕ, calculer les composantes du vecteur normal dans le repère cartésien Réf. Pour chaque valeur de la normale, calculer le vecteur contrainte associé et tracer la position de l'extrémité du vecteur contrainte dans le repère de référence Réf. *Commenter le résultat.*

3. Diagonaliser le tenseur de contraintes Σ (`help eig`). On note $\sigma_I, \sigma_{II}, \sigma_{III}$ les valeurs propres (valeurs diagonales du tenseur D de la fonction `eig`). On note $\overrightarrow{n}_I$, $\overrightarrow{n}_{II}$, $\overrightarrow{n}_{III}$ les vecteurs propres associés. La fonction `eig` donne les composantes des vecteurs propres dans le repère de référence Ref. On rappelle que par définition, le tenseur de contraintes est diagonal dans le repère principal RP :

$$\mathbf{\Sigma}_{RP} = \begin{pmatrix} \sigma_I & 0 & 0 \\ 0 & \sigma_{II} & 0 \\ 0 & 0 & \sigma_{III} \end{pmatrix}$$

et que dans ce même repère principal RP, les composantes des vecteurs propres sont :

$$\overrightarrow{n}_I = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \overrightarrow{n}_{II} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \overrightarrow{n}_{III} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

4. Tracer dans le repère de référence Ref les axes portés par chaque vecteur propre (sur le tracé précédent) et *de commenter le sens physique des vecteurs propres.*
5. Porter la grandeur des valeurs propres sur chaque vecteur propre et *conclure.* On remarquera en particulier que quel que soit le repère, on a :

$$\overrightarrow{\sigma}(\overrightarrow{n}_I) = \sigma_I.\overrightarrow{n}_I \quad \overrightarrow{\sigma}(\overrightarrow{n}_{II}) = \sigma_{II}.\overrightarrow{n}_{II} \quad \overrightarrow{\sigma}(\overrightarrow{n}_{III}) = \sigma_{III}.\overrightarrow{n}_{III}$$

6. On se place maintenant dans le repère principal RP. On note (X, Y, Z) les coordonnées d'un point de $\mathbb{R}^3$ relatif au repère principal RP. Créer une nouvelle fenêtre en la numérotant (`help figure`). Tracer les axes du repère principal RP pour -4<X<4, -4<Y<4, -4<Z<4.
7. Dans le repère principal, on note les composantes du vecteur normal comme suit :

$$\overrightarrow{n}_{RP} = \begin{pmatrix} L \\ M \\ N \end{pmatrix}$$

et le vecteur contrainte associé s'écrit :

$$\overrightarrow{\sigma}(\overrightarrow{n}_{RP}) = \mathbf{\Sigma}_{RP}.\overrightarrow{n}_{RP} = \begin{pmatrix} \sigma_I & 0 & 0 \\ 0 & \sigma_{II} & 0 \\ 0 & 0 & \sigma_{III} \end{pmatrix} \begin{pmatrix} L \\ M \\ N \end{pmatrix} = \begin{pmatrix} \sigma_I L \\ \sigma_{II} M \\ \sigma_{III} N \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Comme précédemment, on fait varier l'angle θ de 0°à 360°tous les 10°et l'angle ϕ de 0°(au pôle nord) à 180°(au pôle sud) tous les 10°aussi. Pour chaque valeur de θ et de ϕ , calculer les composantes du vecteur normal $\overrightarrow{n}_{RP}$ dans le repère principal RP.

8. Pour chaque valeur de la normale $\overrightarrow{n}_{RP}$, calculer le vecteur contrainte associé $\overrightarrow{\sigma}(\overrightarrow{n}_{RP})$ et tracer la position de l'extrémité du vecteur contrainte dans le repère principal RP.
9. Comme $\overrightarrow{n}_{RP}$ est un vecteur de norme 1, on a

$$L^2 + M^2 + N^2 = 1$$

D'où par identification :

$$\frac{X^2}{\sigma_I^2} + \frac{Y^2}{\sigma_{II}^2} + \frac{Z^2}{\sigma_{III}^2} = 1$$

En conclusion, quand la normale à la facette balaie tout l'espace, l'extrémité du vecteur contrainte balaie une surface ellipsoïdale dont les axes sont portés par les vecteurs propres et les longueurs sont données par les valeurs propres. On appelle cette surface l'ellipsoïde de LAME des contraintes.

Utiliser la fonction Matlab `surf` (`help surf`) pour tracer cet ellipsoïde. On utilisera surf(XX,YY,ZZ) avec des matrices XX,YY et ZZ de taille ntheta fois nphi. Commenter le résultat. Conclure.

5.5 Expressions et fonctions MATLAB utilisées

`figure`
`plot3`
`eig`
`surf`

Chapitre 6

TP5 : Application de l'analyse en composantes principales à des données de procédé de traitement biologique en lit bactérien (4 séances TP)

Objectif

Le but de ce TP est de traiter des données réelles d'une campagne de mesure d'effluents industiels à l'aide d'outils statistiques et d'algèbre linéaire. L'objectif de ce traitement de données est de dégager de centaines de mesures des conclusions pertinentes et des solutions industrielles efficaces.

6.1 Méthodologie

- Créer le répertoire $TP5$ qui contiendra tous les scripts et toutes les fonctions à programmer. Nommer `main.m` le script principal qui regroupe les lignes de commandes principales de la programmation du TP.
- Pour trouver de l'aide sur une fonction MATLAB ou rechercher le nom d'une fonction, lancer les commandes `help` et `lookfor` (`help nom_de_la_commande` dans la fenêtre de commande MATLAB. Exemple : `help eig`). Regarder à la fin de l'aide, le paragraphe `See also` pour connaître le nom des fonctions similaires.
- Ajouter commentaires et indentation pour améliorer la lisibilité du code pour un usage ultérieur.
- L'utilitaire d'édition *imposé* estt l'éditeur `Emacs`.
- **Il est fortement conseillé de travailler par écrit sur une feuille (algorithme) avant de commencer l'écriture du programme avec le langage** MATLAB.
- **Il est aussi fortement conseillé de tester les lignes de commandes au fur et à mesure de la construction de votre programme**
- Pensez à écrire un programme principal condensé et à faire appel à des fonctions aussi souvent que possible.
- **Un programme principal commence toujours par les instructions :**
  `clear all` : efface toutes les variables en mémoire
  `clf` : efface toutes les figures

6.2 Introduction au traitement biologique en lit bactérien

6.2.1 Lits bactériens

La difficulté majeure dans la lutte contre la pollution industrielle réside dans :
- la grande variabilité des types de rejets d'un secteur industriel à un autre.
- la diversité des agents polluants rencontrés et donc des paramètres à mesurer.

Les rejets industriels peuvent être pollués par :
- des matières organiques (MO)
- des matières en suspension (MES)
- des toxiques : phénols, cyanures...
- des métaux lourds.

Afin d'épurer les effluents industiels, il existent plusieurs techniques possibles. Concernant les matières organiques, les procédés biologiques permettent le plus souvent d'obtenir la dépollution recherchée dans des conditions économiques acceptables. Le plus courant de

ces procédés, dit par "boues activées" met en œuvre une biomasse libre, maintenue en suspension dans le liquide qu'il a pour objet d'épurer, en utilisant la pollution comme source d'énergie. D'autres techniques sont employées, comme les "filtres bactériens" qui utilisent une biomasse fixée sur un support granulaire.

6.2.2 Systèmes de contrôle

L'efficacité de ces dépollutions nécessite un contrôle régulier. Or les stations d'épuration biologiques sont faiblement pourvues en capteurs et autres appareils de mesure. Ce type d'équipement nécessite la présence de personnels aptes à comprendre et à interpréter les éventuelles variations ainsi que l'assurance d'une maintenance régulière. On peut cependant trouver ponctuellement certaines sondes de type pH ou Red-ox répondant à des besoins spécifiques des usines.

Il apparaît donc nécessaire de proposer des méthodes de contrôle proposant l'utilisation de capteurs, décrivant les caractéristiques physico-chimiques des effluents et permettant ainsi de :

- détecter rapidement les dérives du procédé
- isoler le problème
- établir un diagnostic complet de l'état du système
- mettre en œuvre l'action corrective adéquate

A cette fin, on peut envisager deux types d'approche :

- un diagnostic basé sur des techniques d'intelligence artificielle (lois heuristiques).
- un diagnostic établi à partir de l'historique de fonctionnement du procédé et son analyse statistique. C'est cette deuxième approche qui sera abordée ici.

6.2.3 Description de la campagne de mesures

L'étude proposée porte sur le fonctionnement d'une station d'épuration d'une industrie chimique. Cette dernière produit majoritairement de l'acide-amino-11-décanoïque qui sert, entre autre, à la fabrication de gainage métallique et de rilsans. Ce produit chimique est élaboré à partir de d'huile de ricin, de glycérine et de xylène, composés qui se retrouvent donc naturellement dans les effluents à traiter.

Dans le procédé d'épuration utilisé, les effluents sont séparés en deux catégories : produits non huileux et composés huileux. Ces derniers suivent un premier traitement d'élimination des graisses via une unité de flottation. Ils rejoignent ensuite les produits dits " non huileux " afin de subir une première décantation, dont le rôle est d'enlever les matières en suspension.

L'effluent percole ensuite à l'intérieur du lit bactérien où se réalise le traitement biologique. Le lit bactérien est de forme cylindrique : 12.6 mètres de hauteur et 18 mètres de diamètre. A la sortie du lit bactérien, une deuxième décantation est nécessaire de manière à retenir les boues et autres éléments détachés du filtre bactérien et assurer un rejet d'effluent aux caractéristiques conformes avec la législation en vigueur.

6.2.4 Paramètres mesurés

Le premier type de données que l'on va traiter concerne les caractéristiques physico-chimiques de l'effluent traité. Ces mesures ont été effectuées au moyen de deux sondes multiparamètres immergées, placées en amont et en aval du lit bactérien. Les paramètres mesurés sont :

- La température (°Celsius)
- Le potentiel Red-Ox (mV)
- La conductivité (mS / Cm). La conductivité électrique d'une eau est la conductance d'une colonne d'eau comprise entre deux électrodes métalliques de $1cm^2$ de surface et séparées l'une de l'autre de $1cm$. D'un point de vue pratique, la mesure est basée sur le principe du pont de wheatstone et la cellule de conductimétrie (platine + noir de platine) constitue l'une des branches du pont. De plus, il faut noter que la conductivité est très influencée par la température.
- L'oxygène dissous (mg / L). La sonde utilisée est basée sur le principe polarographique. Cette méthode s'appuie sur la polarisation des électrodes par application d'une tension électrique constante. On va ainsi mesurer les variations de courant provoquées par la réduction de l'oxygène qui a diffusé à travers la membrane.
- Le pH. La mesure du pH, avec une électrode en verre (de référence interne au calomel) est liée à l'activité des ions présents en solution.
- La turbidité (NTU). La sonde utilise le phénomène de TYNDALL, technique dite de néphélométrie : un liquide trouble s'éclaire vivement lorsqu'il est traversé par un faisceau lumineux. Ce sont les particules en suspension qui diffractent la lumière.
- Le pourcentage de saturation en O_2

Tous ces paramètres ont été mesurés pendant une durée de 29 jours, avec un intervalle de mesure de 15 minutes, ce qui correspond à environ 2800 valeurs par paramètres à analyser.

6.3 Introduction à l'analyse en composantes principales (ACP)

6.3.1 Nécéssité d'un traitement statistique

Avant toute utilisation de données issues de mesures chimiques, il faut vérifier leur validité afin d'avoir une bonne représentation de la réalité. Les sources d'interférence étant nombreuses (électromagnétique, problème d'installation, faible maintenance), il faut pouvoir les détecter et traiter les valeurs potentiellement erronées. Chaque série de mesures est souvent confrontée à des problèmes de bruit, valeurs manquantes, points aberrants, de tendance. Les objectifs d'un traitement statistique de ces données sont :

- traiter un grand nombre de données, quelles soient dépendantes ou indépendantes.
- réduire la dimension du problème (le nombre de variables mesurées).
- réaliser des prédictions sur les variables qualitativement importantes à partir des variables des procédés et de variables qualitatives.
- proposer des solutions à la pollution industrielle.

L'étude et la description d'un ensemble de données sont faciles lorsque celles-ci sont représentées par un nombre réduit de paramètres (appelés variables descriptives). En effet,

la représentation graphique de ces variables permet de mettre rapidement en évidence, par simple examen du nuage de points, les relations éventuelles entre variables ou données, mais cette efficacité diminue à mesure que la dimension de la représentation augmente. Ainsi, une fois le tridimensionnel dépassé, il devient impossible de représenter le nuage d'étude.

L'analyse en composantes principales est une méthode de réduction du nombre de variables nécessaires pour représenter géométriquement les phénomènes. La réduction n'est possible que si les N variables initiales ne sont pas indépendantes. Cette notion d'indépendance est mesurée à l'aide des cœfficients de corrélation que nous verrons ensuite. Il faut que les cœfficients de corrélation soient non nuls. L'ACP est une méthode dite factorielle, car la réduction ne consiste pas en une sélection des variables de base mais en une définition de nouvelles variables (principales), obtenues par combinaison des variables initiales. C'est une méthode linéaire. L'outil mathématique associé à la méthode d'ACP repose donc sur l'algèbre linéaire et le calcul matriciel.

6.3.2 Définition géométrique de l'ACP

Considérons un tableau de données contenant les mesures effectuées des différentes variables (ici la température, la turbidité...) aux différents instants (appelés événements). Lorsqu'il n'y a que 2 variables, il est facile de représenter l'ensemble des données sur un graphe à 2 dimensions. Le tracé permet de déduire :

- l'absence de corrélation entre les variables
- l'existence d'une forte liaison
- l'émergence de sous-populations ou de groupes de variables.

Si il y a N variables mesurées et M événements par variable, on recherche une représentation graphique réalisable (c'est-à-dire sur un plan) la plus fidèle à la réalité possible (c'est-à-dire qui minimise les distorsions). Pour cela, on utilise une projection orthogonale du nuage de points sur un plan choisi de manière à rendre maximale la moyenne des carrés des distances entre les variables.

La méthode consiste à rechercher :

- la direction Δ_1 rendant maximum la moyenne des carrés des distances entre variables,
- la direction Δ_2 orthogonale à Δ_1 rendant maximum la moyenne des carrés des distances entre variables,
- les directions Δ_3, Δ_4, Δ_5... orthogonales entre elles.

Ces directions ne sont rien d'autres que les directions principales du nuage de points.

Soit $X_1, X_2, X_3, .., X_N$ les coordonnées de l'ensemble des N variables dans le repère de référence. On note $C_1, C_2, C_3, .., C_N$ les coordonnées de l'ensemble des N variables dans le repère principal. La matrice de changement de repère correspond aux coordonnées des vecteurs principaux (ou vecteurs propres) dans le repère principal.

Remarque : Dans le TP4 d'algèbre linéaire, vous aviez recherché les vecteurs principaux d'un tenseur de contraintes.

6.3.3 Définition algébrique de l'ACP

Mathématiquement, on peut interpréter chaque variable comme un vecteur de $\mathbb{R}^M$ (ici il y a 15 vecteurs de $\mathbb{R}^{2800}$). On note $X_1, X_2, .., X_N$ ces N vecteurs de $\mathbb{R}^M$ (ici $N = 15$ et $M = 2800$). On peut aussi sauvergarder ces données dans une matrice X de taille $M \times N$ dont le cœfficient X_{ij} représente la ième mesure de la jème variable. Les vecteurs colonnes de X sont $X_1, .., X_N$ et pour $j = 1..N$, $X_j \in \mathbb{R}^M$ représente les mesures de la $j$ème variable à tous les $M = 2800$ instants. De même, on note $e_1, .., e_M$ les vecteurs lignes de X et pour $i = 1..M$, $e_i \in \mathbb{R}^N$ représente les mesures à l'instant i des $N = 15$ variables.

On introduit un peu de vocabulaire en définissant pour $j = 1..N$:

– la moyenne $\bar{X}_j$ de la variable X_j

$$\bar{X}_j = \frac{1}{M}\sum_{i=1}^{M} X_{ij}$$

– l'ecart-type σ_j

$$\sigma_j = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(X_{ij} - \bar{X}_j)^2}$$

– la matrice R (carrée de taille $N \times N$) de corrélation des variables $X_1, ..., X_N$

$$R_{jk} = \frac{\sum_{i=1}^{M}(X_{ij} - \bar{X}_j)(X_{ik} - \bar{X}_k)}{\sqrt{\sum_{i=1}^{M}(X_{ij} - \bar{X}_j)^2 \sum_{i=1}^{M}(X_{ik} - \bar{X}_k)^2}} = \frac{\sum_{i=1}^{M}(X_{ij} - \bar{X}_j)(X_{ik} - \bar{X}_k)}{M\sigma_j\sigma_k}$$

Pour plus de facilité, on préfère travailler avec des variables centrées réduites, c'est-à-dire de moyenne nulle et d'écart-type égale à 1. Soit donc ces nouvelles variables $Y_1, ..., Y_N$:

$$\forall 1 \leq i \leq M\,, \forall 1 \leq j \leq N\,, \quad Y_{ij} = \frac{X_{ij} - \bar{X}_j}{\sigma_j}$$

Dans cette transformation, la matrice de corrélation est inchangée, c'est-à-dire que la matrice de corrélation des variables $Y_1, ..., Y_N$ est encore R. En particulier,

$$R_{jk} = \frac{1}{M}\sum_{i=1}^{M} Y_{ij}Y_{ik} = \frac{1}{M}Y^TY$$

Soit V le sous espace vectoriel de $\mathbb{R}^N$ sur lequel on souhaite projeter le nuage de points pour réduire le nombre de variables que l'on considère (souvent V sera un plan) et soit P la projection orthogonale de $\mathbb{R}^N$ sur V. D'après la définition géométrique précédente, le but de l'ACP est de choisir V tel que

$$I_V = \sum_{i=1}^{M}\sum_{k=1}^{M} \|Pe_i - Pe_k\|^2$$

soit maximum où on a noté $\|.\|$ la norme euclidienne dans $\mathbb{R}^N$. On a alors le théorème suivant (qu'on admettra) :

Théorème 1. *La matrice de corrélation R est symétrique donc diagonalisable dans une base orthonormale. On note $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$ ses valeurs propres et $v_1, ..., v_N$ les vecteurs propres associés. Soit q un entier fixé inférieur ou égal à N. La valeur maximale de I_V pour V sous-espace vectoriel de dimension q de $\mathbb{R}^N$ est atteinte pour $V = Vect(v_1, ..., v_q)$ et $I_V = \lambda_1 + ... + \lambda_q$.*

Si $q = N$, on obtient les directions principales du nuage de points. Si $q < N$, on obtient les q variables les plus représentatives pour décrire le nuage de points (On retrouve que ces variables ne sont pas une sélection des variables de base mais des nouvelles variables (principales) obtenues comme combinaison linéaire des variables originales).

On note C_j et Z_j les composantes principales des N variables X_j ou Y_j respectivement. On sait (voir TP4 le chapitre sur les rappels mathématiques) que les vecteurs C_j et Z_j sont donnés en fonction de X_j et Y_j respectivement par N relations linéaires qui traduisent le changement de repère :

$$C^T = PX^T \Leftrightarrow C_k = \sum_{j=1}^{N} P_{kj} X_j \qquad Z^T = PY^T \Leftrightarrow Z_k = \sum_{j=1}^{N} P_{kj} Y_j$$

On vérifie que

$$\sum_{j=1}^{N} P_{ij} P_{kj} = \delta_{ik} \text{ cad } P^T P = I_N$$

ce qui traduit bien le fait que P est la matrice de changement de repère entre deux bases orthonormales.

Les variables Z_j sont de moyenne nulle et de variance λ_j :

$$\lambda_j = \frac{1}{M} \sum_{i=1}^{M} Z_{ij}^2$$

Enfin, une propriété remarquable des valeurs propres est que leur somme est égale à la dimension du problème :

$$\sum_{j=1}^{N} \lambda_j = N$$

On connait ainsi la contribution en variance de chaque composante principale à la variance totale du sytème.

6.4 Programmation

6.4.1 Réception et tracé des données de bases

On fournit deux fichiers issus de tableur excel sous format csv : dse.csv et dss.csv relatifs aux données de sondes en entrée et en sortie du lit bactérien. Le fichier dse.csv a 2866 lignes (du 02/06/01 à 15 heures au 03/08/01 à 11h15) et 8 colonnes (température °C, condsp mS/cm, conductivité mS/cm, oxygène dissous saturé pourcentage, oxygène dissous

pourcentage, pH, redox mV, turbidité NTU). Le fichier dss.csv a 2875 lignes (du 02/06/01 à 15 heures au 03/08/01 à 13h30) et 7 colonnes (température °C, condsp mS/cm, conductivité mS/cm, oxygène dissous saturé pourcentage, oxygène dissous pourcentage, pH, redox mV).

1. Créer un fichier script.m et charger ces fichiers après les avoir copiés depuis le répertoire `/home/commetud/2eme\ Annee\ ICBE/Informatique/TP_ACP/` dans votre répertoire de travail. (voir Remarque ci-après pour importer des fichiers au format `csv` sous Matlab)
2. Tracer 4 par 4 les évolutions temporelles des données d'entrée et de sortie (`help subplot`)

6.4.2 Calcul des variables centrées réduites

3. Pour chaque variable X_j, calculer la moyenne $\bar{X}_j$ et l'écart-type (`help mean` et `help std`).
4. Définir les variables centrées réduites Y associées au 15 variables initiales X.
5. Vérifier que la moyenne de chaque variable centrée réduite Y est nulle et que l'écart-type est égale à 1.
6. Calculer la matrice de corrélation R à partir des variables en coordonnées réduites.

6.4.3 Composantes principales et vecteurs principaux

Composantes principales

7. Diagonaliser la matrice R (`help eig`).
8. En déduire les composantes principales λ_j.
9. Vérifier la propriété remarquable des valeurs propres : $\sum_{j=1}^{N} \lambda_j = N$. Déterminer ainsi la contribution en variance de chaque composante principale à la variance totale du système.
10. Faire un tracé des valeurs propres en fonction de leur indice. Combien de variance expliquent les premières valeurs propres ?

Directions principales

Les coordonnées des vecteurs propres sont définies dans le repère initial, relatif aux N variables initiales.

11. Construire un tableau à N+1 colonnes (relatives aux N vecteurs propres et à la liste des variables initiales) et N lignes relatives aux variables initiales (et donc aux coordonnées de chaque vecteur propre dans le repère initial).
12. Pour chaque vecteur propre, identifier les cœfficients les plus importants et identifier les variables initiales associées.

Interprétation des composantes principales

La principale question de l'ACP réside dans la signification concrète des composantes principales qui représentent les nouvelles variables obtenues par combinaison linéaire des variables initiales. On étudie en général les corrélations avec les variables initiales en terme de cercle de corrélation.

Pour cela, on calcule les cœfficients de corrélation entre la kième composante principale C_k et la jième variable initiale X_j : c'est la jième composante du kième vecteur propre v_k multipliée par la racine de λ_k :

$$r(C_k, X_j) = \sqrt{\lambda_k}(v_k)_j$$

Soit (C_i, C_k) un couple de variables principales contribuant pour beaucoup à la variance totale. Dans un plan (ici $q = 2$), on représente chaque variable X_j par un point d'abscisse $r(C_i, X_j)$ et d'ordonnée $r(C_k, X_j)$. On obtient alors un nuage de N points correspondant aux N variables initiales projetées sur le plan $Vect(v_i, v_k)$. On peut montrer que tous ces points sont situés dans le cercle unité centré à l'origine appelé cercle de correlation et que plus un point de coordonnées $(r(C_i, X_j), r(C_k, X_j))$ est près du cercle, plus la corrélation est effective entre la variable d'origine X_j et les variables principales C_i et C_k.

13. Définir un vecteur abs correspondant aux coordonnées du mième vecteur principal multipliées par la racine de la valeur propre ou composante principale λ_m associée.
14. Définir un vecteur coor correspondant aux coordonnées du nième vecteur principal multipliées par la racine de la valeur propre ou composante principale λ_n associée.
15. Tracer les couples de points (abs,coor) dans le repère relatif à C_m en abscisse et C_n en ordonnées.
16. Tracer le cercle de rayon unité relatif au cercle de corrélation.
17. Identifier les variables discriminées par C_m et celles discriminées par C_n.

Effectuer ces tracés pour les couples (m, n) de composantes principales d'importance décroissante, en terme de contribution à la variance.

6.5 Expressions et fonctions MATLAB utilisées

```
help
lookfor
eig
std
mean
plot
sqrt
diag
figure
hold on
axis
size
```

Chapitre 7

TP6 : Application des séries de Fourier à l'étude du violon (2 séances TD, 2 séances libres et 4 séances TP)

Objectif

L'objectif de ce TP est de créer un semblant de violon numérique en faisant jouer des sons de "violon" à l'ordinateur. Ce TP comporte des travaux théoriques de modélisation du mouvement d'une corde de violon et des travaux numériques de programmation de ces résultats.

Quelques conseils :

- Créer le répertoire $TP6$ qui contiendra tous les scripts et toutes les fonctions à programmer.
- Pour écrire ces fichiers vous utiliserez un éditeur de texte. Les utilitaires d'édition *imposés* sont l'éditeur propre à MATLAB ou `Emacs`.
- **Il est fortement conseillé de travailler par écrit sur une feuille (algorithme) avant de commencer l'écriture du programme avec le langage** MATLAB. L'objectif est de lister et d'organiser les tâches à réaliser séquentiellement.
- **Il est aussi fortement conseillé de tester les lignes de commandes au fur et à mesure de la construction de votre programme**. Pour cela vous pouvez copier une ou plusieurs lignes de commande et les coller dans la fenêtre MATLAB. Si vous utilisez l'éditeur de texte associé à MATLAB : sélectionnez des lignes de commandes, clic-droit puis `evaluate selection` (version complète nécessaire) ce qui déclenche l'exécution des lignes en question dans la fenêtre MATLAB.
- Pour trouver de l'aide sur une fonction MATLAB ou rechercher le nom d'une fonction, lancer les commandes `help` et `lookfor` (`help nom_de_la_commande` dans la fenêtre de commande MATLAB. Exemple : `help lookfor`).
- Regarder à la fin de l'aide d'une fonction, le paragraphe `See also` pour connaître le nom des fonctions similaires.
- Ajouter commentaires et indentation pour améliorer la lisibilité de vos codes. Cela vous fera gagner beaucoup de temps lors d'un usage ultérieur.

7.1 Etude du mouvement de la corde de violon

On cherche à représenter mathématiquement le mouvement d'une corde de violon frottée par un archet. On schématise donc la corde à l'instant $t \geq 0$ par une courbe $y = f(t, x)$ dans un plan $(x0y)$. On la suppose de longueur L au repos, attachée à ses deux extrémités $(0, 0)$ et $(0, L)$. Une fois que l'archet a donné son impulsion, on suppose la corde libre et sans amortissement.

Cette corde possède les propriétés physiques suivantes :

- une masse par unité de longueur $\mu = 6\,10^{-4} kg.m^{-1}$,
- une tension $\mathcal{T}\rceil = 200\,N$ réglée lors de l'accord du violon par le musicien,
- une longueur $L = 0.5\,m$.

La relation fondamentale de la dynamique permet alors d'écrire que

$$\partial_t^2 f(t,x) = c^2 \partial_x^2 f(t,x) \quad x \in [0, L] \quad t \geq 0 \tag{7.1}$$

où $c = \sqrt{\mathcal{T}\rceil/\mu} > 0$.

On associe à cette équation aux dérivées partielles les conditions aux limites et conditions initiales :

$$\forall t \geq 0\,, \qquad f(t,0) = f(t,L) = 0 \tag{7.2}$$

$$\forall x \in [0,L]\,, \quad f(0,x) = 0 \quad \partial_t f(0,x) = \alpha(L-x) \tag{7.3}$$

où $\alpha = 0.1$.

7.1.1 Résolution par onde progressive

On pose $u = x - ct$ et $v = x + ct$ et on définit l'application linéaire ϕ de la façon suivante :

$$\begin{array}{rccl} \phi: & [0,L]\times\mathbb{R}^+ & \to & \phi([0,L]\times\mathbb{R}^+) \\ & (x,t) & \mapsto & (u,v) = (x-ct, x+ct) \end{array}$$

Enfin, on définit la fonction F par

$$f(t,x) = F(x-ct, x+ct) = F(u,v)\,.$$

1. Montrer que ϕ est un difféomorphisme de classe C^2. Calculer $\Omega = \phi([0,L]\times\mathbb{R}^+)$.
2. Exprimer $\partial_x^2 f(t,x)$ et $\partial_t^2 f(t,x)$ en fonction de $\partial_u^2 F(u,v)$, $\partial_v^2 F(u,v)$, $\partial_{uv}^2 F(u,v)$.
3. Si f est solution de (7.1), trouver l'équation aux dérivées partielles satisfaite par F.
4. Montrer qu'il existe alors deux fonctions h et g de classe C^2 sur $\mathbb{R}$ telles que
$$F(u,v) = h(u) + g(v)\,.$$
5. En déduire que la solution de (7.1) s'écrit
$$f(t,x) = h(x-ct) + g(x+ct)\,, \quad \forall (x,t) \in [0,L]\times\mathbb{R}^+$$
6. Trouver h et g pour que f soit solution de (7.2) et (7.3). Montrer que f est de la forme
$$f(t,x) = \frac{\alpha}{4c}\left((L-|x-ct|)^2 - (L-|x+ct|)^2\right)$$
si $t \in [-\frac{T}{2}, +\frac{T}{2}]$, $x \in [0,L]$ avec $T = \frac{2L}{c}$ et f est T-périodique.

7. *Créer trois fonctions* `solexacte_h`, `solexacte_g` *et* `solexacte` *qui pour un réel t (représentant le temps) donné et un vecteur* `x` *de points (de la corde) calcule respectivement $h(x-ct)$, $g(x+ct)$ et $f(t,x) = h(x-ct) + g(x+ct)$. L'entête de l'une de ces fonctions pourra s'écrire*

```
function g = solexacte_g(t,x)
%Fonction qui calcule la composante g(x+ct) de la
%solution exacte du mouvement de la corde de violon
%pour un temps t et aux points contenus dans x.
%Le vecteur g est de meme dimension que x.
% VARIABLES GLOBALES UTILISEES :
% Alpha, c, L.
% Appel a la fonction :
% gexaxte = solexacte_g(temps,abscisses);
```

On prendra un pas de discrétisation en espace de 1 mm. On pourra utiliser la fonction `rem` *pour replacer le temps sur l'intervalle* $[-T/2, T/2]$ *(*`help rem`*). Les paramètres* α, c, L *seront déclarées en variables globales. Vous testerez vos fonctions en tracant ces fonctions pour* $t = 0, T/4, T/2, 3T/4, T$*. Commentez les courbes obtenues.*

8. *Dans un fichier script et à l'aide des fonctions* `getframe` *et* `movie` *(s'aider de l'exemple donné dans l'aide de* `getframe`*) , créer trois animations montrant l'évolution de* h *(en rouge), de* g *(en bleu) puis de* f *(en noir). On pourra enfin superposer l'évolution des trois fonctions sur une même et dernière animation. On pourra prendre un pas de temps de* `T/100` *et on observera les animations sur un intervalle de temps de* T*. On fixera le nombre d'images par seconde (ou Frames Per Second : paramètre* `FPS` *de* `movie`*) à 100 et on fixera les axes comme suit :*

```
axis([0 0.5 -alpha/(16*c) alpha/(16*c)])
```

Remarque : La fonction $(t, x) \to g(x + ct)$ est constante le long de la droite $x + ct = constante$ appelée caractéristique. Elle représente une onde progressive se propageant sur l'axe des x à la vitesse c de la droite vers la gauche. De même, la fonction $(t, x) \to h(x - ct)$ est constante le long des caractéristiques $x - ct = constante$. Elle représente une onde progressive se propageant à la même vitesse c mais de la gauche vers la droite.

7.1.2 Résolution par méthode de séparation des variables

On souhaite maintenant résoudre l'équation des ondes (7.1) grâce à la méthode de séparation des variables.

1. On cherche une solution élémentaire de l'équation des ondes de la forme

$$f(t, x) = \phi(x)\psi(t)\,, \quad \forall (x, t) \in [0, L] \times \mathbb{R}^+$$

vérifiant les conditions aux limites (7.2). Montrer que cette solution élémentaire peut s'écrire :

$$\mathcal{F}_n(x, t) = \Big(A_n \cos(2\pi f_n t) + B_n \sin(2\pi f_n t)\Big) \sin(k_n x)\,, \quad n \in \mathbb{N}\,, \tag{7.4}$$

où A_n et B_n sont des constantes et

$$f_n = \frac{nc}{2L}\,, \qquad k_n = \frac{n\pi}{L}\,; \tag{7.5}$$

f_n est appelée fréquence de la nième harmonique.

2. Quel lien existe-t-il entre f_1 et f_n ? Il s'ensuit que le son du violon est périodique de période $1/f_1$ (dans notre modèle idéal !). Ceci est encore vrai pour beaucoup d'autres sons ! f_1 est appelé le fondamental.
3. Montrer que la solution générale de l'équation des ondes s'écrit :

$$f(t, x) = \sum_{n=1}^{+\infty} (A_n \cos(2\pi f_n t) + B_n \sin(2\pi f_n t)) \sin(k_n x)\,.$$

4. Trouver les A_n et B_n pour que f vérifie (7.3).

5. En déduire que l'unique solution de (7.1, 7.2, 7.3) s'écrit

$$f(t,x) = \sum_{n=1}^{+\infty} \frac{2\alpha L^2}{n^2\pi^2 c} \sin(k_n x) \sin(2\pi f_n t)\,. \tag{7.6}$$

6. On veut maintenant programmer la solution $f(t,x)$ donnée par (7.6), le problème étant que l'ordinateur ne sait pas calculer une somme infinie !
   On va donc tronquer la série et ne garder que les p premiers termes :

$$u_p(t,x) = \sum_{n=1}^{p} \frac{2\alpha L^2}{n^2\pi^2 c} \sin(k_n x) \sin(2\pi f_n t).$$

   Montrer à l'aide d'une majoration très grossière de comparaison avec une intégrale que l'erreur commise est inférieure à $\frac{2\alpha L^2}{\pi^2 cp}$ $\forall (t,x)$.

7. *Créer une fonction* `solapprochee(t,x,p)` *qui calcule* $u_p(t,x)$.

8. *On va fixer* $t = T/3$. *Tracer à l'aide de la commande* `subplot` $u_p(t,x)$ *pour* $p = \{1,3,10,100\}$ *et comparer (sur les mêmes graphiques mais avec une autre couleur) les courbes à la solution exacte calculée à l'aide de la fonction* `solexacte`.

9. *Dans cette question aussi on fixe* $t = T/3$. *Tracer sur un même graphique et en fonction de* p *l'erreur commise (*`max(abs(solapprochee(t,x,p)-solexacte(t,x)))`*) ainsi que la majoration calculée dans la question précédente. On pourra utiliser une échelle logarithmique (*`help loglog`*) et faire varier* p *de 1 à 100.*

7.2 Son produit par l'instrument

Le son produit par le violon peut être approximé par la force exercée sur le chevalet par la corde. Ce chevalet est relié à la table des harmoniques d'où la note est émise, et correspond à la frontière de notre modèle située en L. Cette force est assimilée à la dérivée spatiale du mouvement de la corde en L. On définit donc $s(t)$ le son émis par le violon au court du temps par :

$$s(t) = \frac{\partial}{\partial x} f(x,t)\Big|_{x=L} = \sum_{n=1}^{+\infty} d_n sin(2\pi f_n t) \tag{7.7}$$

On définira v_p une approximation du son émis par l'instrument à un instant t, la quantité :

$$v_p(t) = \sum_{n=1}^{p} d_n sin(2\pi f_n t) \tag{7.8}$$

10. Calculer d_n et vérifier que $|d_n| \sim 1/n$ quand n tend vers $+\infty$. Ceci traduit le fait que l'amplitude des harmoniques successives décroît doucement avec n, et donc que le violon est un instrument riche en harmoniques aiguës. *Définir une fonction Matlab* `dn` *permettant le calcul des coefficients* d_n *de la relation (7.7). Tracer un histogramme des 30 premiers éléments de la suite* $(d_n)_{n\in\mathbb{N}}$ *(*`help bar`*).*

11. *Définir une fonction* `vp` *calculant l'approximation du son pour un temps donné. Tracer la courbe représentant le son sur l'intervalle* $[0, T]$. *Construire le son émis par le violon sur un intervalle de temps de deux secondes. Quelle est la fréquence d'échantillonage (sample frequency : paramètre Fs de soundsc) de ce signal. Jouer ce son (*`help soundsc`*).*

12. On souhaite rendre le son un peu plus réaliste. Pour celà on introduit un temps de montée et d'atténuation de la note. On obtient ces effets en multipliant l'amplitude du signal sonore par une enveloppe qui vous est donnée.
Copier la fonction `envelop.m` *depuis le répertoire*
`/home/commetud/2eme\ Annee\ ICBE/Informatique/TPson/` *dans votre répertoire de travail :*
`cp /home/commetud/2eme\ Annee\ ICBE/Informatique/TPson/envelop.m .`
Construire un vecteur représentant le profil de montée et d'atténuation (`help envelop`*). Représenter graphiquement l'enveloppe. Construire le son avec la montée et l'atténuation, le visualiser et le jouer.*

7.3 Expressions et fonctions Matlab utilisées

```
help
lookfor
function
global
rem
getframe
movie
axis
subplot
abs
loglog
bar
soundsc
envelop
```

Chapitre 8

Annexe : lexique non exhaustif

Caractères spéciaux et opérateurs logiques	
Symbole	**Usage**
=	instruction d'affectation
()	utilisées pour marquer la priorité d'une expression arithmétique ; contiennent les variables en entrée des fonctions
[ ]	utilisé pour former matrices et vecteurs ; contient les variables en sortie des fonctions
.	point décimal
..	répertoire au-dessus du répertoire courant
...	continue une suite d'instructions à la ligne suivante
;	termine une ligne ; evite l'affichage
,	sépare les éléments ou instructions d'une ligne et les variables d'une fonction
%	commentaires
:	utilisés pour la génération de vecteurs lignes d'éléments régulièrement espacés, par défaut avec un incrément entier, sinon avec un incrément à préciser ; désigne l'ensemble des indices d'un tableau selon une de ses dimensions
(:)	force le format vecteur colonne
'	se place avant et après une chaine de caractères
''	permet de définir ' à l'intérieur d'une chaine de caractères
@	identificateur de fonction depuis la version 6
!	permet l'execution d'une commande du système d'exploitation
&	ET logique ; opère élément par élément pour les tableaux avec "faux" valant 0 et toute valeur non nulle valant "vrai"
|	OU logique
~	NON logique
`xor`	OU exclusif, s'emploie sous la forme `xor(A, B)`
`any`	fonction vectorielle logique qui vaut 1 si un élément du vecteur est non nul et 0 sinon ; elle opère sur chaque colonne pour une matrice et s'emploie sous la forme `any(A)`
`all`	fonction vectorielle logique qui vaut 1 si chaque élément du vecteur est non nul ; s'emploie et opère comme la précédente pour les tableaux

Opérateurs arithmétiques

Symbole	Usage
+	addition
-	soustraction
*	produit matriciel ; les dimensions doivent être compatibles (sauf le cas d'un scalaire)
.*	produit élément par élément ; les deux variables doivent avoir les mêmes dimensions
\	division à gauche, ou solution de système(s) linéaire(s) : `C = A \ B` si `A*C=B`
.\	division à gauche, élément par élément
/	division à droite, ou solution de système(s) linéaire(s) : `C = B/A` si `C*A=B`
./	division à droite, élément par élément
^	puissance de scalaire ou de matrice
.^	puissance de chaque élément d'un tableau
'	conjuguée transposée
.'	transposée

Opérateurs relationnels

Symbole	Usage
<	inférieur, compare élément par élément des tableaux de même taille et prend pour valeur un tableau de cette taille , de coefficients 1 ("vrai") ou 0 ("faux")
<=	inférieur ou égal, opére comme le précédent
>	supérieur
>=	supérieur ou égal
==	égal
~=	différent

Variables et fonctions d'intérêt général

Symbole	Usage
`help`	aide en ligne
`helpdesk`	accès à une documentation hypertexte de Matlab, gérée par Netscape sous Unix
`what`	fournit la liste des M-fichiers du repertoire courant
`type`	affiche le contenu d'un M-fichier
`lookfor`	indique les occurences d'une chaine de caractères dans l'aide en ligne
`ans`	résultat de la dernière instruction lorsqu'il n'est pas affecté
`whos`	affiche la liste des variables courantes, et leur format
`save`	sauvegarde une ou plusieurs variables de la session dans un fichier du répertoire courant
`load`	retrouve les variables sauvegardées précédemment
`clear`	supprime une ou plusieurs variables de la session

Variables et fonctions d'intérêt général	
Nom	**Usage**
`function`	définit une fonction ; en pratique, on enregistre les instructions qui la définissent dans un M-fichier de même nom que la fonction (muni du suffixe `.m`)
`feval`	évalue une fonction dont le nom est donné en argument
`nargin`	nombre de variables en entrée d'une fonction
`nargout`	nombre d'arguments en sortie d'une fonction
`varargin`	liste d'arguments variable en entrée
`varargout`	liste d'arguments variable en sortie ; définit une cellule de tableaux
`global`	définit des variables globales
`pause`	arrête la session en attente d'une réponse de l'utilisateur
`disp`	affiche une variable sans donner son nom, ou un texte
`find`	fournit les indices des éléments non nuls d'un tableau
`for`	initialise une boucle de répétition d'une suite d'instructions pour des valeurs d'une variable (compteur) spécifiées par un vecteur
`while`	initialise une boucle de répétition d'une suite d'instructions tant qu'une condition reste vraie
`if`	instructions éxécutées sous condition
`else`	s'utilise avec `if`
`elseif`	s'utilise avec `if`
`end`	clôt le corps d'instructions de `for`, `while` et `if`
`break`	arrête l'éxécution des boucles `for` ou `while`
`size`	dimensions d'un tableau
`length`	longueur d'un vecteur
`linspace`	crée un vecteur de composantes uniformément réparties entre deux valeurs
`logspace`	crée un vecteur de composantes réparties logarithmiquement entre deux valeurs
`fliplr`	inverse l'ordre des colonnes d'un tableau
`flipud`	inverse l'ordre des lignes d'un tableau
`reshape`	change les dimensions d'un tableau (avec les mêmes éléments)
`repmat`	crée un tableau en reproduisant une matrice selon les dimensions spécifiées
`cat`	crée une matrice par concatenation
`cell`	crée une cellule de tableaux
`struct`	crée une structure de matrices
`format`	précise le format d'affichage
`echo`	controle l'affichage des commandes éxécutées
`more`	controle le nombre de lignes de chaque page affichée
`tic`	ouvre le compteur de durée d'execution d'une suite de commandes
`toc`	ferme le compteur ouvert par `tic`
`cputime`	compteur de temps CPU
`cd`	change le repertoire de travail
`quit`	termine une session Matlab

Arithmétique, polynômes et manipulation de données	
Nom	**Usage**
`eps`	précision relative de l'arithmétique virgule-flottante utilisée
`pi`	π
`i,j`	unité imaginaire des nombres complexes
`abs`	valeur absolue ou module d'un nombre complexe
`angle`	argument d'un nombre complexe
`sqrt`	racine carrée
`real`	partie réelle
`imag`	partie imaginaire
`conj`	conjugué complexe
`gcp`	plus grand diviseur commun
`lcm`	plus petit multiple commun
`round`	arrondi à l'entier le plus proche
`fix`	arrondi vers 0
`ceil`	arrondi vers ∞
`floor`	arrondi vers $-\infty$
`rem`	reste après division entière : `rem(x,y)= x-fix(x./y).*y`
`mod`	reste signé : `mod(x,y)=x-floor(x./y).*y`
`max`	plus grande composante d'un vecteur ; pour une matrice, la fonction renvoie un vecteur ligne formé des maxima de chaque colonne
`min`	plus petite composante d'un vecteur
`sort`	trie les composantes d'un vecteur selon l'ordre croissant
`mean`	moyenne des composantes d'un vecteur
`sum`	somme des composantes d'un vecteur
`std`	écart-type des composantes d'un vecteur
`cov`	matrice de covariance
`prod`	produit des composantes d'un vecteur
`cumsum`	sommes cumulées des éléments d'une matrice selon une dimension à préciser
`poly`	construit un polynôme de racines données
`roots`	fournit les racines d'un polynômes par recherche des valeurs propres de la matrice compagne
`polyval`	evalue un polynôme donné par ses coefficients
`conv`	produit de polynômes, ou convolution de vecteurs
`deconv`	division polynômiale ou déconvolution de vecteurs
`polyder`	fournit le polynôme dérivé
`fzero`	résoud une équation non linéaire
`fsolve`	résoud un système d'équations non linéaires
`fmin`	recherche le minimum d'une fonction d'une variable
`fmins`	recherche le minimum d'une fonction de plusieurs variables
`diff`	opérateur aux différences ; permet de construire les différénces finies ou divisées

Algèbre linéaire	
Nom	**Usage**
`zeros`	matrice de 0
`ones`	matrice de 1
`eye`	matrice identité
`rand`	matrice aléatoire, de coefficients uniformément distribués sur $[0, 1]$
`randn`	matrice aléatoire, de coefficients normalement distribués
`diag`	extrait la diagonale d'une matrice, ou crée une matrice diagonale ou bande
`tril`	extrait la partie triangulaire inférieure d'une matrice
`triu`	extrait la partie triangulaire supérieure d'une matrice
`inv`	inverse d'une matrice (par résolution de systèmes linéaires)
`lu`	factorisation lu d'une matrice, avec stratégie de pivot partiel
`chol`	factorisation de Choleski d'une matrice symétrique définie positive
`qr`	factorisation orthogonale-triangulaire d'une matrice
`null`	fournit une base orthonormée du noyau
`orth`	fournit une base orthonormée de l'image
`rank`	rang d'une matrice
`eig`	tous les éléments propres d'une matrice carrée
`svd`	valeurs et vecteurs singuliers d'une matrice
`pinv`	pseudo-inverse d'une matrice
`det`	déterminant
`cond`	nombre de conditionnement pour la résolution des systèmes linéaires, en norme 2
`condest`	estimation du conditionnement d'une matrice carrée, en norme 1
`norm`	norme d'une matrice (plusieurs normes sont proposées)
`trace`	trace d'une matrice
`expm`	exponentielle de matrice
`sqrtm`	racine carrée matricielle d'une matrice carrée

Analyse	
Nom	**Usage**
`exp, log, log10`	exponentielle de base e, logarithme néperien et de base 10
`sin, asin, sinh, asinh`	sinus, arcsinus, sinus hyperbolique et argsh
`cos, acos, cosh, acosh`	cosinus,...
`tan, atan, tanh, atanh`	tangente,...
`cot, acot, coth, acoth`	cotangente,...
`sec, asec, sech, asech`	secante,...
`csc, acsc, csch, acsch`	cosecante,...
`besselj, bessely, besselh`	fonctions de Bessel de première, deuxième et troisième espèce
`besseli, besselk`	fonctions de Bessel modifiées
`gamma, gammainc`	fonctions gamma
`beta, betainc`	fonctions beta
`erf, erfinv`	fonction d'erreur (ou fonction de Gauss) et sa réciproque

Graphiques et visualisations	
Nom	**Usage**
`figure`	crée une nouvelle fenêtre de visualisation
`plot`	graphe linéaire en dimension 2
`fplot`	graphe d'une fonction entre deux valeurs
`bar`	graphe en rectangles verticaux
`hist`	histograme
`pie`	graphe en camembert
`polar`	graphe en coordonnées polaires
`subplot`	permet de partitionner une fenêtre de visualisation
`hold`	conserve le graphique courant
`axis`	définit les bornes des axes
`title`	affiche un titre (chaine de caractères à fournir)
`legend`	affiche une légende sur la figure
`text`	affiche un commentaire (chaine de caractères à fournir) en un point donné par ses coordonnées
`xlabel`	précise le nom de la variable en absisse
`semylogy`	graphique avec une echelle logarithmique en ordonnée
`plot3`	graphe linéaire en dimension 3
`bar3`	graphe en parallèlogrames verticaux
`meshgrid`	construit un jeu de coordonnées pour la visualisation tridimensionnelle d'une fonction de 2 variables
`mesh`	visualise une surface maillée en dimension 3
`hidden`	fait apparaitre (`off`) ou disparaitre (`on`) les parties cachées d'une surface maillée
`surf`	visualise une surface ombrée (en couleurs) en dimension 3
`surfnorm`	représente les normales aux surfaces en dimension 3
`contour`	représente les isolignes d'une surface en dimension 2
`meshc`	combine `mesh` et `contour`
`clabel`	fait apparaitre les valeurs des isolignes
`quiver`	représente des gradients par des flèches
`contour3`	représente les isolignes d'une surface en dimension 3
`view`	spécifie le point de vue d'un graphe en dimension 3
`colormap`	définit le jeu de couleurs
`colorbar`	affiche une échelle des couleurs
`getframe`	forme un vecteur colonne à partir d'un graphique en vue d'une animation
`movie`	execute l'animation visuelle avec la matrice précédente
`image`	crée une image en interpretant les valeurs des coefficients d'une matrice
`print`	imprime un graphique, ou le sauvegarde en format post-script

Analyse numérique	
Nom	**Usage**
`spline`	spline cubique d'interpolation
`interp1`	interpolation de données en dimension 1
`interpn`	interpolation de données en dimension n
`quad`	intégration numérique par la méthode de Simpson
`quadl`	intégration numérique par une methode adaptative de Lobatto
`ode45`	résolution approchée d'équations ou de systèmes différentiels non raides par méthodes de Runge-Kutta emboitées d'ordre 4-5
`ode113`	résolution approchée d'équations ou de systèmes différentiels non raides par des méthodes d'Adams-Bashforth-Moulton de type PECE d'ordre variable
`ode23s`	résolution approchée d'équations ou de systèmes différentiels raides par la méthode de Rosenbrock d'ordre 2
`ode15s`	résolution approchée d'équations ou de systèmes différentiels raides par une méthode de différentiation rétrograde d'ordre variable
`fft`	transformée de Fourier rapide en dimension 1
`ifft`	transformée de Fourier rapide inverse en dimension 1
`fft2`	transformée de Fourier rapide en dimension 2

Matrices creuses	
Nom	**Usage**
`sparfun`	fournit une liste de fonctions qui s'appliquent aux matrices creuses
`sparse`	convertit au format creux une matrice pleine, ou crée une matrice creuse
`full`	convertit au format plein une matrice creuse
`speye`	matrice identité en format creux
`spdiags`	crée une matrice diagonale ou une matrice bande en format creux
`nnz`	nombre d'éléments non nuls
`issparse`	vrai si la matrice est sous format creux
`spy`	visualise la répartition des coefficients non nuls d'une matrice
`symmmd`	renumérotation par l'algorithme du degré minimum
`symrcm`	renumérotation par l'algorithme Cuthill et Mac-Kee inverse
`pcg`	résoud un système linéaire de matrice symétrique définie positive par la méthode du gradient conjugué préconditionné
`gmres`	résolution d'un système linéaire par la méthode GMRES
`luinc`	factorisation LU incomplète d'une M-matrice creuse
`cholinc`	factorisation de Choleski incomplète d'une H-matrice creuse
`eigs`	quelques éléments propres d'une matrice carrée

Chapitre 9

Index des fonctions et expressions MATLAB utilisées

Chapitre 10

Eléments de programmation structurée :

10.1 Une démarche structurée pas-à-pas :

La démarche structurée peut être vue comme découlant naturellement d'une démarche scientifique puisqu'une partie de celle-ci a vocation à développer des modèles qui structurent les connaissances. On peut donc tenter de décrire succinctement les étapes qui permettent d'aboutir à une solution programmée à partir d'un problème d'ingénierie :

1. Réflexion scientifique : description du problème scientifique ou technique à résoudre
2. Réflexion mathématique : analyse du problème, choix d'outil et algorithmique
3. Action sur papier : développement d'une stratégie de résolution
   liste de tâches → organigramme → pseudo code
4. Action programmation : écriture du programme, exécution et analyse des résultats

10.2 Quelques idées de base de la programmation structurée :

L'idée de base est de construire de manière structurée un programme qui permet de résoudre un problème complexe en le décomposant en un ensemble de sous-problèmes plus simples (= sous-programmes ou fonctions). La simplification se fait par niveau d'analyse descendante jusqu'à arriver à une description par modules les plus simples possibles qui permettent une programmation aisée tout en évitant les astuces de programmation (caractère générique).
La décomposition d'un programme en modules permet

- d'affiner sa perception
- de faciliter le travail en le simplifiant (travail en groupe, échelonné dans le temps)
- de vérifier le fonctionnement d'un niveau indépendamment de l'ensemble du programme

Un programme structuré suit une architecture très générale de type :

- Début du programme (arguments d'entrées/sorties)
- Instructions déclaratives

– Instructions exécutables
– Fin de programme

10.3 Organigramme en programmation structurée :

La structuration des programmes et sous-programmes facilite la présentation sous forme d'organigramme. Pour en faciliter la compréhension, une syntaxe graphique type est proposée (voir page suivante)

10.4 Programmation structurée avec Matlab :

Quelques idées de structures sous Matlab :
L'idée de base est de construire un programme principal aussi appelé script (par exemple main.m) qui comprend l'essentiel :

– commenter l'objet du programme
– définir les variables globales du problème (les autres sont locales par défaut)
– traiter le problème par une liste d'instructions connues de matlab ou définies dans des sous-programmes (fichiers 'function')
– sortir et présenter les résultats

Tout ce qui n'est pas essentiel à un premier niveau d'interprétation doit si possible être développé dans des fichiers 'function'. Si la fonction le nécessite, elle peut elle-même appeler une autre fonction, toujours dans le but de structurer et simplifier au maximum la structure de chaque fonction :

– commenter l'objet de la fonction, les arguments d'entrée et de sortie
– définir les variables globales utilisées par la fonction (et seulement celles utiles)
– traiter le problème par une liste d'instructions connues de matlab ou définies dans des sous-programmes (fichiers 'functions')
– revenir au niveau précédant en fournissant les arguments de sortie (s'ils existent) ou les variables globales nouvellement affectées suite aux instructions de la fonction.

Un exemple simple :
Le fichier principal.m contient le texte suivant

```
%Ce programme principal permet de donner un exemple de structure
%pour calculer et représenter Fa(x)=sin(a*x)
%J'efface les figures et toutes les variables en mémoire pour éviter toute confusion
clear figure;
clear all;
%Déclaration des variables globales (gardées en mémoire)
global a
%Affectation des variables
a=3 ;
x=[0 :pi/100 :pi] ;
%Appel à la fonction Fa.m avec l'argument d'entrée 'x'
y=Fx(x);
%Visualisation
```

```
plot(x,y);
%Fin du programme
```

Le fichier Fx.m contient le texte suivant :

```
function arg_sortie=Fx(arg_entree);
% Cette fonction permet de calculer F(x)=sin(a*arg_entree)
% ' arg_entree ' est un argument d'entrée et 'a' est une variable globale
% Rappel des variables globales en mémoire utiles ici:
global a;
% Calcul de Fx avec l'argument d'entrée 'arg_entree
arg_sortie =sin(a*arg_entree);
% Retour au programme appelant avec l'argument de sortie ' arg_sortie '
```

Dans cet exemple, la variable `x` est dite *locale* pour le programme `principal.m`. La valeur de la variable `x` est affectée à l'argument d'entrée `arg_entree` de la fonction `Fx.m`. Cette variable `arg_entree` est locale pour la fonction `Fx.m` (x n'existe pas pour `Fx.m`). De même, la variable locale `arg_sortie` de la fonction `Fx.m`) est un argument de sortie affecté à la variable locale `y` du programme `principal.m` (`arg_sortie` n'existe pas pour `principal.m`). La seule variable connue et partagée par les 2 fichiers est la variable globale `a`.

Chapitre 11

Bibliographie

Matlab Guide, D. J. Higham, N. J. Higham, SIAM, 2000.

Introduction to Scientific Computing, A Matrix-Vector Approach Using MATLAB, C.F. Van Loan, MATLAB curriculum Series, 1997.

Apprendre et Maîtriser Matlab, versions 4 et 5 et SIMULINK^r, M. Mokhtari, A. Mesbah, Springer, 1997.