

TRAVAUX PRATIQUES DE L'UV MT31

MATHÉMATIQUES : APPLICATIONS

Automne 2004

Jérôme BASTIEN

Document compilé le 15 mars 2005

Liste des Travaux Pratiques

Avertissement	3
Travaux Pratiques 1. Connaissances de base de matlab	5
1.1. «Cahier des charges» : objectifs à atteindre	5
1.2. Lancement et instructions de base	7
1.3. Les notions de bases	8
1.4. Le répertoire courant et le chemin d'accès	9
1.5. La programmation avec matlab	10
1.6. Résolution de systèmes linéaires	12
1.7. Valeurs et vecteurs propres	13
1.8. Les complexes	14
1.9. Les polynômes	14
1.10. Les graphiques	15
1.11. L'utilisation du debugger	15
1.12. Matlab symbolique	15
1.13. Quelques exemples en matlab	17
Travaux Pratiques 2. Dérivées et gradients	23
2.1. Initiation au matlab symbolique	23
2.2. Quelques calculs de dérivées symboliques	23
2.3. Étude de fonctions de \mathbb{R}^2 dans \mathbb{R} et tracé des isovaleurs et du gradient	24
Travaux Pratiques 3. Intégrales doubles	29
Travaux Pratiques 4. Un problème de rectangles	31
4.1. Introduction	31
4.2. Résultats théoriques	32
4.3. Partie pratique	36
4.4. Exemples de simulations numériques	37
Bibliographie	43

Avertissement

Le TP 1 d'introduction à matlab est en partie extrait du livre [BM03] (pages 335 à 347). Il constitue l'annexe B de cet ouvrage et a été reproduit avec l'aimable autorisation des auteurs.

Les notions présentées correspondent à la version 6 de matlab mais elles sont standard et ne devraient guère évoluer au cours des versions ultérieures de matlab.

Les TP 3 et 4 sont facultatifs et pourront être traités éventuellement par ceux qui auraient un peu d'avance.

Connaissances de base de matlab

La sections 1.1 constituent un «cahier des charges» qui précisent les objectifs à atteindre.

Les sections 1.2, 1.3, 1.4, 1.5, 1.6 et 1.7 sont extraites¹ de [BM03]. Elles constituent le cœur de ce TP. Quelques sources matlab sont évoquées dans ces sections. Toutes ces sources sont disponibles sur <http://utbmjb.chez.tiscali.fr/>

La section 1.13 est un ensemble d'exemples², qui sera traité de façon facultative.

1.1. «Cahier des charges» : objectifs à atteindre

1.1.1. Organisation : écran et espace de travail

1.1.1.1. *Ecran*. Les étudiants feront apparaître au minimum trois fenêtres :

- une pour les commandes matlab ;
- une pour l'édition des fichiers .m en cours d'écriture ;
- une pour les représentations graphiques connexes.

1.1.1.2. *Espace de travail*.

- Les étudiants définiront l'arborescence nécessaire à une présentation claire de leurs travaux pratiques.
- Ils apprendront à s'y déplacer, à utiliser les commandes associées le plus souvent issues d'Unix.
- Les règles de visibilité des fichiers écrits devront être rapidement maîtrisées.
- Par défaut, les passages de paramètres de fonctions se font par valeur ; il est possible de globaliser une variable entre les différents blocs dont on souhaite qu'ils la partagent.

1.1.2. Outils disponibles

1.1.2.1. *L'organisation de matlab à l'UTBM*. matlab est constitué d'un noyau et de toolboxes spécifiques (par exemple splines, symbolique, optimisation etc...), utilisatrices de ce noyau. L'UTBM dispose, pour des raisons de droit, de deux versions du logiciel fournis par des serveurs différents :

- Version «enseignement»
 - Pour la version «enseignement» accessible depuis l'ensemble des postes de l'établissement, un serveur dévolu gère les jetons disponibles, tant au niveau du noyau, que des toolboxes satellites.
 - Un jeton est «pris» par l'utilisateur dès qu'il appelle une fonction appartenant à la boîte concernée.
 - Il est restitué dès que l'utilisateur quitte le logiciel, mais pas avant ; par conséquent, sortez du logiciel pour ne pas handicaper les autres utilisateurs éventuels.
- Version «recherche dans le cadre de contrats»

¹Elles en constituent l'annexe B.

²Tous extraits des sujets de TD.

- Une version «recherche dans le cadre de contrats» est aussi accessible et fonctionne sur le même principe. Elle est pour l’instant moins riche en toolboxes, en raison de leur coût plus élevé.
- Le fournisseur nous interdit en effet d’utiliser la version «enseignement» dans le cadre de travaux de recherche donnant lieu à des contrats. Une utilisation de matlab ne respectant pas cette règle est frauduleuse et par suite punissable pénalement.

1.1.2.2. *Les fonctions disponibles.*

- Il convient d’apprendre rapidement à utiliser l’aide afin de connaître les fonctions disponibles sur votre poste, à une date donnée. Ceci vous évitera de réécrire des fonctions poussives à la place de celles existantes...
- Pour les fonctions existantes vous apprendrez à faire apparaître les commentaires fournis dans les sources ; pour les fonctions que vous définirez vous-mêmes, vous devrez apprendre à écrire des commentaires de façon convenable. Voir en particulier l’utilisation de : **help nom_de_fonction**, **lookfor mot_clé**.

1.1.3. Prise en main du logiciel

Il convient absolument de maîtriser les aspects suivants ; les trois premiers seront abordés en découverte de matlab, le dernier sera approfondi par les étudiants eux-mêmes.

1.1.3.1. *Calculs ordinaires dans matlab.* Après quelques explications sur les variables de l’espace de travail, après avoir vu comment les sauver et les rappeler, les étudiants découvriront l’essentiel des possibilités ordinaires sur les vecteurs, matrices, etc...

1.1.3.2. *Les possibilités graphiques.* Les rudiments relatifs aux représentations devront être maîtrisés. Les plus avancés pourront se plonger dans la définition des objets graphiques, de leurs enfants, des handles. Voir fonctions **get** et **set**.

1.1.3.3. *L’écriture de fonctions : les fichiers.m.*

- Tout fichier développé est considéré comme une fonction qui pourra être utilisée ultérieurement dans un autre cadre applicatif. Ceci a le grand avantage de forcer l’utilisateur à une pensée claire, puisqu’il se voit contraint de préciser ce qui est champ d’entrée, de sortie et variable locale. Les éventuelles variables globales, réduites au nombre minimal, seront déclarées globales dans tous les fichiers qui les partagent. Impérativement, l’utilisateur documente ses fonctions ou ses scripts dans les lignes (contigües) qui suivent immédiatement sa déclaration. Ainsi lors d’un appel de help «nom de la fonction écrite» l’utilisateur verra apparaître la définition des champs de la fonction appelée.
- Au moment de construire une démo finale, on pourra écrire un script matlab, contenant des saisies de valeurs par l’utilisateur.
- Il importe de profiter de la réalisation de travaux pratiques sous matlab pour apprendre les spécificités d’un logiciel et pour l’utiliser au mieux, en prenant en compte ses qualités et ses défauts. Il est plus que regrettable d’utiliser matlab comme le C par exemple. En conséquence, on évitera sous matlab, au maximum, le recours aux for en particulier, très lents dans ce cadre, car souvent remplaçables par des produits matriciels, très performants car ils font partie du domaine d’excellence de matlab. Ainsi utiliser des for sous matlab est plus qu’une faute de goût..., c’est presque une ineptie. L’utilisateur réfléchira à la façon dont il peut avoir recours à des outils performants. Dans les applications lourdes, lorsqu’on est contraint de faire appel à

des blocs itératifs, on les écrit sous C ou Fortran et on appelle les routines depuis matlab qui prend en charge les liens : voir les fichiers .mex.

1.1.3.4. *Le symbolique.* Le calcul symbolique de matlab (ou calcul formel) fonctionne de façon similaire à maple³. Voir section 1.12.

L'objet de cette annexe est de rappeler de façon succincte au lecteur les notions élémentaires de matlab à partir de quelques exemples, utilisés dans certains TP.

Le lecteur ne connaissant pas matlab pourra consulter des guides spécialisés (par exemple [HLR01], [Ref92], [PES99], [MM97] et [Mok00])

Nous indiquons à chaque fois les séquences à taper sous matlab.

1.2. Lancement et instructions de base

- **help** : très précieuse, cette aide en ligne vous permet de «tout savoir sur tout». Pour savoir comment l'utiliser, faites **help help**. Cette aide est dite en ligne, puisqu'on l'utilise directement dans la fenêtre de commande matlab. Vous pouvez aussi lancer l'aide de matlab, en cliquant **help matlab** dans le menu help de la fenêtre de commande matlab ; cette aide est plus conviviale, parfois mieux documentée. De surcroît, elle contient parfois des références pour comprendre le fonctionnement des algorithmes.
- **lookfor xxx** : affiche⁴ le nom des toutes les fonctions qui contiennent le mot xxx (en anglais!) dans la première ligne de commentaire. C'est *grosso modo* la fonction réciproque de **help**.

Nous soulignons l'importance des fonctions **help** (ou de la fenêtre d'aide) et **lookfor** ; l'utilisateur de matlab n'hésitera pas à les utiliser avant de programmer une fonction.

- **demo** : démonstration de matlab, très complète, dans laquelle on trouvera des exemples variés, couvrant l'ensemble des domaines d'utilisation.
- **intro** : lance une introduction à matlab.
- **cd** : permet de changer de répertoire.
- **pwd** : affiche le répertoire courant.
- **dir** ou **ls** : affiche la liste des fichiers du répertoire courant.
- **delete** : supprime un fichier.
- ... : pour terminer une expression à la ligne suivante.
- ↑ : fait passer à la commande précédente et permet, par exemple, de la modifier sans la retaper.
- ↓ : fait passer à la commande suivante.

À partir de la fenêtre de matlab, on peut exécuter des commandes matlab :

- les unes après les autres, comme une «super calculatrice», qui ferait un nombre de choses incroyable,
- ou groupées sous forme de fichiers script ou de fonctions (cf. section 1.5.3).

Dans les deux cas, on tape seulement une commande par ligne, ou plusieurs séparées par des point-virgules. Si on tape seulement l'instruction, le résultat apparaît juste après ; si on rajoute un point-virgule à la fin de la ligne, la commande est exécutée mais son résultat n'apparaît pas.

³En fait, matlab utilise les fonctions de maple.

⁴Attention, cette fonction est très lente, puisqu'elle consulte l'ensemble des fichiers .m de matlab

1.3. Les notions de bases

1.3.1. Les variables

Elles sont d'un type unique (matrice) et n'exigent aucune déclaration. On pourra, pour simplifier, distinguer les scalaires (réels ou complexes ; cf. section 1.8), les vecteurs et les matrices.

1.3.2. Les opérations arithmétiques

Elles sont les mêmes que sur les calculatrices : $+$, $-$, $/$, $*$, $(,)$, $^$ (pour des puissances entières ou réelles) dotées des règles de priorité classiques.

1.3.3. Vecteurs ou tableaux à une dimension

Il y a plusieurs façons de saisir un vecteur, par exemple :

```
x=[6 5 6 ]
x=[6, 5, 6 ]
```

Ce vecteur est considéré comme une matrice à une ligne et trois colonnes. On pourra le vérifier en tapant :

```
size(x)
[m,n]=size(x)
length(x)
```

On peut concaténer deux vecteurs lignes :

```
v=[x 1 2 3]
w=[1 x 2 3]
```

On récupère les composantes d'un vecteur en spécifiant son indice entre parenthèses :

```
v(2)
```

Pour obtenir un vecteur dont les composantes varient à pas constant entre deux valeurs connues, on utilisera

```
x=0:0.25:1
y=-pi:0.3:pi
z=0:10
```

Si on impose le nombre de valeurs, on utilisera

```
t=linspace(-pi,pi,4)
```

En général, on privilégiera l'emploi de `:` plutôt que de `linspace`, plus lent d'emploi.

Les opérations sur les vecteurs à n composantes sont naturellement celles de l'espace vectoriel $(\mathcal{M}_{n,p}(\mathbb{R}), +, \cdot)$: on additionne les matrices composante par composante et on les multiplie par un scalaire en multipliant chacune de leurs composantes par ce scalaire.

En faisant précéder d'un point les opérateurs $*$, $/$, et $^$, on peut effectuer d'autres opérations élément par élément :

```
x.*y
x.^2
x./y
x.^y
```

De même, en utilisant toutes les fonctions classiques (sin, exp, cos, sqrt,...) de matlab, on calcule l'image de chaque composante.

Il existe un grand nombre de fonctions matlab opérant directement sur les vecteurs. Citons par exemple **sum** et **prod**.

Chaque fonction matlab que vous programmerez sera de préférence matricielle. Comme les fonctions matlab prédéfinies, elles doivent, si possible, opérer sur un tableau avec la règle suivante : l'image d'un tableau par une fonction est le tableau des images.

1.3.4. Vecteurs ou tableaux à deux dimensions

Saisie d'une matrice :

```
a = [1 2 3 ; 6 7 8 ]
```

Voici un certain nombre d'opérations sur les matrices; à vous de les utiliser et d'en saisir le fonctionnement.

```
a = [1 2 3; 4 5 6 ; 7 8 0]
```

```
b=a'
```

```
c=a+b
```

```
a*b
```

```
a^3
```

```
exp(a)
```

```
poly(a)
```

```
det(a)
```

```
magic(6)
```

```
a=magic(10)
```

```
a(1:5,3)
```

```
a(1:5,7:10)
```

```
a(:,3)
```

```
a(1:5,:)
```

```
a(1,[1 3 5])
```

```
a(:,10:-1:1)
```

```
a(:)
```

```
a(:)=1:100
```

Pour ces huit dernières opérations sur les matrices, vous pouvez consulter l'aide à la rubrique colon.

1.4. Le répertoire courant et le chemin d'accès

Le répertoire courant (current directory) est le répertoire où l'on travaille (cela est important si on utilise des fichiers stockés sur disque dur). Pour le modifier, cliquer sur les trois petits points de la fenêtre de commande (à côté de **current directory**) et choisir le répertoire souhaité.

Par ailleurs, il est possible de rajouter au chemin d'accès (path) un certain nombre de répertoires ou de sous-répertoires où matlab viendra chercher les fichiers *.m recherchés (pour les éditer, les faire fonctionner, consulter leur commentaire). Pour cela, cliquer **set path** dans le menu **file**, puis **add folder** ou **add folder with subfolder**. On aura intérêt à mettre en fin de path (avec l'instruction **move bottom**) les répertoires rajoutés de façon que matlab consulte tout d'abord ses propres répertoires avant les vôtres.

Dans le path, figurent en effet par défaut les différents sous-répertoires de matlab. Grâce au path, si vous utilisez **edit**, **help**, **lookfor** ou si vous lancez une exécution, matlab ira consulter l'ensemble des fichiers des répertoires présents dans le path, même si vous travaillez dans un autre répertoire courant.

1.5. La programmation avec matlab

1.5.1. les fonctions et les M-fichiers

Dans les exemples précédents, vous avez utilisé un certain nombre de fonctions de matlab. Par exemple, **det**(a) renvoie le déterminant de la matrice a : c'est une fonction à une variable, qui pour toute matrice carrée renvoie son déterminant.

On peut définir soi-même ses propres fonctions. On veut définir la fonction de \mathbb{R}^3 dans \mathbb{R}^2 qui à toute matrice 1×3 , $a=(x, y, z)$ associe la matrice 1×2 , $b=(u, v)$ où

$$\begin{cases} u = \sqrt{x^4 + y^6} - z, \\ v = xyz. \end{cases}$$

On lance l'éditeur matlab (à partir de la fenêtre matlab, on choisit **file/new**) ou on tape **edit** dans la fenêtre de commande ; avec cet éditeur, on tape :

```
function res=dudu(a)
res(1)=sqrt((a(1))^4+(a(2))^6)-a(3);
res(2)=a(1)*a(2)*a(3);
```

On enregistre ce fichier avec le nom dudu (l'extension est .m par défaut). On tape dans la fenêtre de commande matlab :

```
a=[1 4 5 ];
dudu(a)
```

Il est indispensable que le nom d'une fonction matlab soit identique au nom du fichier dans laquelle elle est stockée.

EXERCICE 1.1. Soit la fonction de \mathbb{R} dans \mathbb{R} , définie par morceaux par :

$$f(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 & \text{si } 0 \leq x \leq 2, \\ 2 & \text{si } 2 < x < 18, \\ 3 & \text{si } x \geq 18. \end{cases}$$

Écrire cette fonction sous matlab de façon matricielle, c'est-à-dire, de telle sorte que l'image d'un tableau soit le tableau des images. On pourra consulter la fonction fournie **escalier**.

Une autre possibilité permet de définir une fonction sans passer par un fichier (on réservera cette solution à une fonction simple et utilisée un nombre restreint de fois) en utilisant la fonction **inline** très pratique, dont voici un exemple d'utilisation :

```
g=inline('x.^2+sin(x)');
```

Pour calculer $g(2)$, il suffira de taper, comme pour les fonctions usuelles :

```
g(2)
```

Cette fonction est en mémoire vive (le vérifier en tapant **whos**) et le demeure tant qu'on ne fait pas **clear** et qu'on reste dans la même session de matlab. On pourra consulter l'aide à propos de cette fonction.

On créera ses propres fonctions, chacune d'elle étant enregistrée dans un fichier xxx.m où xxx est le nom de cette fonction ou déclarée en inline. Si on en définit plusieurs, on testera chacune d'elle indépendamment des autres.

Les variables utilisées localement dans une fonction ne sont reconnues que dans cette fonction. Si on veut utiliser des variables globales, on les définira par **global** ; mais on évitera le plus possible le recours à ces variables globales. Dans une fonction, les variables d'entrée ne sont pas modifiées au cours de l'exécution de la fonction.

1.5.2. La programmation

Aux M-fichiers de fonctions se rajoutent les M-fichiers de programmes, dits aussi script ; plutôt que de taper les instructions les unes après les autres dans la fenêtre matlab, il peut être plus agréable de les mettre toutes dans un M-fichier de commande, qui sera édité à partir de l'éditeur et lancé à partir de la fenêtre matlab, en tapant **nomfichier** ou bien **run nomfichier** où nomfichier est le nom du M-fichier (sans l'extension .m).

Bien entendu, matlab autorise le récursif : une fonction peut s'appeler elle-même. Parfois, les informaticiens sont réticents à utiliser la récursivité (trop grand nombre d'appels récursifs de la fonction, non-contrôlabilité de la fin de la boucle).

1.5.2.1. *Les entrées/sorties.* L'instruction **disp(a)** affiche le contenu de la matrice a, qu'elle soit à coefficients scalaires ou de type chaîne.

Pour saisir une variable, on utilise **input** : si on tape

```
x = input('entrez x : ');
```

la variable x peut être : un scalaire, une matrice, une expression algébrique, une expression logique ou symbolique (voir section 1.12).

1.5.2.2. *Les instructions conditionnelles.* Citons les instructions **if**, **end**, **else**, **switch**, **case**, **while**.

1.5.2.3. *Les instructions inconditionnelles : les boucles.* Citons les instructions **for**, **end**.

1.5.3. Différence entre fonction et script

Rappelons les quelques points suivants :

- *Fonction* Une fonction possède des arguments d'entrée, et pas nécessairement de sortie. Elle commence systématiquement par la ligne de commande :

```
function [x,y,z,...] = nom_fonction(a,b,c,...)
```

où x,y,z... sont d'éventuelles variables de sortie et a,b,c... sont les variables d'entrée.

Toutes les fonctions de calculs matlab fonctionnent ainsi ; voir par exemple **det**, **eig**, **diff** ou **int** (en symbolique, voir section 1.12). On pourra aussi consulter les exemples fournis **somme1** ou **escalier**.

Le formalisme adopté dans les TP, pour présenter les fonctions sera le suivant :

```
[x,y,z,...] = nom_fonction(a,b,c,...)
```

- on décrira les variables d'entrée a,b,c, ... : leur nom, leur nombre, leur type
- on décrira les éventuelles variables de sortie x,y,z, ... : leur nom, leur nombre, leur type

On utilisera la fonction décrite ci-dessus en écrivant dans la fenêtre de commande matlab :

```
[ x , y , z ] = nom_fonction ( a , b , c )
```

- *Script* Un script est un fichier de commandes sans variables d'entrée ni de sortie. En revanche, dans un script, l'utilisateur peut saisir des variables en utilisant par exemple la fonction **input**.

Le formalisme adopté dans les TP, pour présenter succinctement les scripts sera le suivant :

nom_script

- descriptif des variables saisies par l'utilisateur ;
- descriptif des actions exécutées par ce script.

On pourra aussi consulter la commande **demo** de matlab, qui est un script, ou des exemples fournis : **demo_exemple_un** ou **demo_exemple_formell**.

On utilisera le script décrit ci-dessus en écrivant dans la fenêtre de commande matlab :

```
nom_script
```

- *Stockage des scripts et des fonctions* Les fonctions et les scripts constituent tous les deux des fichiers *.m. La fonction **nom_fonction** doit être stockée dans un fichier nommé nom_fonction.m.

Pour les TP, la plupart des fichiers à écrire sont des fonctions. Quelques scripts seront demandés : nous les nommerons alors **demo_***.

1.5.4. Quelques conseils

Un très grand nombre de choses sont déjà programmées sous matlab et il serait inutile de les reprogrammer ; bien entendu, il serait excessif de vouloir connaître tout matlab, mais avant de vous lancer dans la programmation, consultez bien les documentations et/ou l'aide.

Utilisez les boucles le moins possible. Souvent on peut faire des calculs sur les composantes d'une matrice sans avoir à accéder à ses éléments. De nombreux calculs en boucles peuvent être remplacés par une interprétation matricielle, plus efficace sous matlab.

Faites du beau travail : décomposez le plus possible votre algorithme de façon à en programmer des petits blocs, indépendants les uns des autres. Dans l'idéal, si vous avez un programme à écrire, il devrait comporter un seul et court M-fichier principal, où l'on définit les valeurs des variables utilisées ; on récupère en sortie, les différents objets calculés ; ce programme fait appel à un certain nombre de fonctions, chacune d'elle en appelant éventuellement d'autres.

Documentez vos programmes : n'hésitez pas à écrire des commentaires. De plus, les lignes commentées qui suivent immédiatement la première ligne de la déclaration d'une fonction apparaîtront dans l'aide. Cela est très précieux et vous permet d'avoir des renseignements sur les fonctions que vous avez définies sans en visualiser les sources, en utilisant **help**. De plus **lookfor** cherchera dans ces lignes (uniquement dans la première, qui doit donc contenir en quelques mots un descriptif sommaire mais précis de la fonction). Les lignes de commentaires séparées du commentaire par une ligne blanche seront ignorées par les fonctions **help** et **lookfor** et permettront à l'utilisateur de commenter son programme.

1.6. Résolution de systèmes linéaires

Pour matlab, il existe deux divisions : / et \ (division droite et gauche) :

$$a/b = ab^{-1} \text{ et } a \backslash b = a^{-1}b.$$

Ainsi, pour des scalaires x et y ,

$$x/y = \frac{x}{y} \text{ et } y \setminus x = \frac{x}{y}.$$

Pour une matrice carrée a , $\mathbf{inv}(a)$ calcule l'inverse de a . Ainsi :

$$a/b = a \mathbf{inv}(b) \text{ et } a \setminus b = \mathbf{inv}(a) b.$$

Cependant, les deux calculs n'utilisent pas les mêmes algorithmes ; en effet, pour résoudre le système linéaire $ax = b$, il est plus rapide d'en calculer sa solution directement que d'abord calculer l'inverse de a et de le multiplier ensuite par b . On étudiera les exemples proposés ci-dessous.

Si B est une matrice carrée, $X = A \setminus B$ est la solution $A^{-1}B$ de l'équation matricielle $AX = B$ où X est de la même taille que B . En particulier, si $B = b$ est un vecteur colonne, alors $x = A \setminus b$ est la solution $A^{-1}b$ de l'équation matricielle $Ax = b$.

Traiter les cinq exemples suivants :

$A = [1 \ 2 \ ; \ 3 \ 4]$;

$B = [5 \ 6 \ ; \ 7 \ 8 \]$;

$C = B / A$

$D = A \setminus B$

$C * A$

$A * D$

$B * \mathbf{inv}(A)$

$\mathbf{inv}(A) * B$

$a = [1 \ 2 \ 3 \ ; \ 4 \ 5 \ 6 \ ; \ 5 \ 7 \ 10]$;

$\mathbf{inv}(a)$

$a = [1 \ 3 \ 5 \ ; \ 1 \ 2 \ 4 \ ; \ 0 \ 5 \ 1]$;

$b = [22; 17; 13]$

$a \setminus b$

$a * (a \setminus b)$

tic ; **for** $k = 1:1000$; $x = \mathbf{inv}(a) * b$; **end** ; $t = \mathbf{toc}$;

disp(t) ;

tic ; **for** $k = 1:1000$; $x = a \setminus b$; **end** ; $t = \mathbf{toc}$;

disp(t) ;

Pour les deux derniers exemples, **tic** lance un chronomètre et **toc** l'arrête et en lit la valeur. Ces deux commandes permettent, ici, de comparer le temps mis par matlab pour calculer $\mathbf{inv}(a) * b$ et $a \setminus b$. Qu'en conclure ?

1.7. Valeurs et vecteurs propres

La commande suivante permet de calculer les valeurs et vecteurs propres de la matrice carrée A :

$[X, D] = \mathbf{eig}(A)$

Ces valeurs et vecteurs peuvent être complexes (cf section 1.8). Il existe un grand nombre d'outils dédiés à l'analyse spectrale de matrices ; nous renvoyons à la bibliographie.

1.8. Les complexes

Matlab accepte les nombres complexes sous la forme

$a+ib$

ou

$\rho * \exp(i \theta)$

On peut aussi noter j à la place de i .

Toutes les opérations algébriques sur les complexes sont reconnues ($*$, $+$, $/$). On dispose aussi des fonctions **real**, **imag**, **abs**, **angle**, **conj**.

On peut élever un complexe à une puissance réelle grâce à la fonction $^{\wedge}$. On dispose aussi de **sqrt**, **log**, **exp**.

Toutes les opérations définies pour les matrices à coefficients réels demeurent valables pour les matrices à coefficients complexes.

Attention, ne pas utiliser un compteur noté i ou j dans un programme où l'on utilise des complexes.

1.9. Les polynômes

Le polynôme

$$P(X) = a_n X^n + a_{n-1} X^{n-1} + \dots + a_1 X + a_0,$$

est représenté en matlab par le vecteur ligne $[a_n \ a_{n-1} \dots \ a_1 \ a_0]$ (de longueur $n + 1$).

Outre les opérations sur les matrices qui s'appliquent donc aux polynômes, il existe un certain nombre de fonctions spécifiques aux polynômes.

- **conv**(A,B) multiplie les deux polynômes A et B.
- $[Q,R]=\mathbf{deconv}(A,B)$ renvoie les deux polynômes Q et R tels que $A = BQ + R$ avec degré de R strictement inférieur à celui de B.
- **roots**(P) renvoie les racines du polynôme P (dans \mathbb{C}). Réciproquement, **poly** calcule le polynôme à partir de ces racines.
- **polyval**(f,X) renvoie les valeurs de la fonction f sur la matrice X.
- **polyder**(f) dérive le polynôme f.
- $[q,d]=\mathbf{polyder}(a,b)$ calcule la dérivée de a/b (où a et b sont deux polynômes) sous la forme q/d .
- si A est une matrice et f un polynôme, **polyval**(f,A) renvoie un tableau constitué de l'image de chacun des éléments de A.

Traiter l'exemple suivant :

```
P=[1 -6 11 -6];
S=roots(P)
poly(S)
poly(S')
```

1.10. Les graphiques

1.10.1. Les graphiques bi-dimensionnels

Rappelons que (et de façon non exhaustive !) :

- **plot(x)** représente les points de coordonnées (j, x_j) , pour $1 \leq j \leq p$, où p est la longueur du vecteur x .
- **plot(x,y)** représente les points de coordonnées (x_j, y_j) , pour $1 \leq j \leq p$.
- **plot(Z)** représente les points de coordonnées $(\text{re}(Z), \text{im}(Z))$, si Z est un vecteur complexe.
- **plot(x1,y1,str1,x2,y2,str2,...,xn,yn,strn)** permet de tracer n courbes (x_i, y_i) avec les paramètres optionnels $\text{str1}, \dots, \text{strn}$.

fplot('fct',lim,str) est très pratique : elle permet de tracer le graphe de la fonction `fct` (M-file ou fonction interne) dans la fenêtre définie par $\text{lim}=[x_{\min}, x_{\max}]$ ou $\text{lim}=[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$; `str` un paramètre optionnel.

On pourra aussi utiliser **ezplot**.

1.10.2. Les graphiques tri-dimensionnels

On étudiera par exemple les fonctions **plot3**, **mesh** ou **surf**.

1.10.3. D'immenses possibilités

Les possibilités de matlab sont immenses et méritent d'être approfondies ...

1.11. L'utilisation du debugger

Avec matlab, il est possible de debugger agréablement ses sources, en suivant l'évolution de l'exécution d'une fonction ou d'un script, pas à pas, ou en s'arrêtant à des endroits précis, tout en contrôlant ou modifiant les différentes variables mises en jeu.

Pour cela, une fois que l'on a édité (avec la fonction **edit**) les différentes fonctions et sources, il faut placer, en déplaçant le curseur, des "points d'arrêt" là où l'on désire voir le programme s'arrêter provisoirement.

On lance ensuite le programme à contrôler depuis la fenêtre de command matlab ; il apparaît une invite "K". On contrôle ensuite l'exécution du programme dans la fenêtre d'édition de plusieurs façons, en appuyant sur les touches F5, F10 ou F11 (voir dans le menu **debugg** leur signification).

Dans tous les cas, apparaît une petite flèche dans l'éditeur qui indique à quel endroit du programme on se trouve. Parallèlement, dans la fenêtre matlab, on peut contrôler les variables et même les modifier.

Quand le programme a tourné ou est en train de tourner avec le debugger, on peut accéder aux valeurs des variables en déplaçant la souris sur chacune des variables dans l'éditeur matlab : sa valeur apparaît alors.

1.12. Matlab symbolique

Nous parlons aussi de matlab formel.

Nous ne donnons ici que quelques exemples très simples. Il convient de consulter l'aide ou les démos pour avoir un regard beaucoup plus complet sur la partie symbolique de matlab, très riche.

On pourra par exemple consulter l'aide pour étudier les fonctions **int**, **diff** et **limit**. Traiter les exemples suivants :

```

syms x a b;
y=x^2+ax+b;
disp(y);
diff(y,x)
diff(y,a)

    ou

syms x h;
limit((sin(x+h)-sin(x))/h,h,0)

    ou

int(sin(x)/x,0,inf)

```

On pourra aussi étudier les fonctions **sym**, **syms**, **solve**, **expand**, **simplify**, **simple**, **subs**, **subexp**, et **pretty**.

Traiter encore les deux exemples suivants qui déterminent de façon symbolique les maximums des fonctions $x \mapsto (a-x)(x-b)$ et $x \mapsto (h-x)x(x+h)$ définies respectivement sur $[a, b]$ et $[-h, h]$, avec tracé des fonctions. Voir les deux scripts **demo_exemple_formel1** et **demo_exemple_formel2** :

% exemple de calcul formel avec trace.

```

syms a b x;
f=(a-x)*(x-b);
fp=diff(f,x);
d=solve(fp);
pretty(simplify(subs(f,x,d)));

disp('pour voir le graphe, appuyez sur une touche');
pause;
X=-1:2/500:1;
Y=subs(subs(f,{a,b},{-1,1}),x,X);
plot(X,Y);

```

% exemple de calcul formel avec trace.

```

syms h x;
f=(h-x)*x*(x+h);
fp=diff(f,x);
d=solve(expand(fp));
q=simplify(subs(f,'x',d));
r=eval(vpa(q));
pretty(eval(vpa(q)));

disp('pour voir le graphe, appuyez sur une touche');
pause;
X=-1:2/500:1;
Y=subs(subs(f,h,1),x,X);

```

```
plot (X,Y);
```

1.13. Quelques exemples en matlab

Dans cette section, nous donnons quelques exemples en matlab : les lignes présentées dans ce TP pourront être tapées et exécutées les unes à la suite des autres, ou mieux, entrées dans script, qui est exécuté ensuite.

Quelques exemples seront issus de TD de cette UV.

1.13.1. Dérivées

On renvoie à la section 1.12 ainsi qu'à la section 2.2 du TP 2.

1.13.2. Intégrales

Grâce au calcul symbolique, on peut calculer des intégrales simples comme :

$$I = \int_0^{\pi/2} \cos^2(x)dx, \quad J = \int_0^1 \sqrt{\frac{ax+1}{x+3}}dx, \quad (a \text{ appartient à } \mathbb{R}_+^*)$$

en tapant

```
syms x a;
I=int (( cos (x) ^ 2),0, pi / 2);
disp (I);
J=int ( sqrt (( a*x+1)/(x+3)),x,0,1);
disp (J);
```

Pour mieux visualiser la valeur de J , on peut aussi remplacer la dernière ligne par `pretty (J);`

On peut aussi calculer des intégrales impropres comme

$$K = \int_0^{\infty} e^{-x^2} dx,$$

en tapant

```
syms x
K=int ( exp(-x ^ 2),0, inf );
```

Comme les mathématiciens, matlab ne sera pas toujours déterminer les primitives des fonctions, comme le montrent

```
syms x
disp ( int ( exp(-x ^ 2),1, inf ) );
disp ( int ( exp(-x ^ 3.5),1,2) );
```

REMARQUE 1.2. On notera la différence des résultats produits par les deux lignes précédentes ; la première d'entre elles renvoie un résultat exprimé à partir de la fonction matlab `erf`. Cette fonction est une fonction qui est connue par les mathématiciens, qu'on peut utiliser sous matlab, en symbolique et en numérique, mais dont on ne dispose pas d'expression explicite exprimées avec des fonctions usuelles⁵. Si on tape dans la fenêtre de commande matlab :

⁵exactement comme la fonction logarithme par exemple.

help erf

on apprend la définition de cette fonction :

$$\forall x \in \mathbb{R}, \quad \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

La seconde ligne de l'exemple donné ci-dessus indique en revanche que l'intégrale demandée

$$\int_1^2 e^{-x^{3,5}} dx,$$

n'est pas connue, même à l'aide de fonctions auxiliaire comme erf.

1.13.3. Matrices

On reprend les exemples a) et b) de l'exercice 3.3 du TD 3 : résoudre les systèmes suivants

$$\text{a) } \begin{cases} 2x + y = 3, \\ 2x + 4y = 6. \end{cases} \quad \text{b) } \begin{cases} x + y + z = 6, \\ 2x - 3y + 4z = 8, \\ x + y - z = 0. \end{cases}$$

Sous matlab, on écrira

```
A=[2 1;2 4];
B=[3;6];
X1=A\B;
C=[1 1 1 ;2 -3 4;1 1 -1];
D=( [6 8 0] )';
X2=C\D;
```

On peut aussi calculer des inverses de matrices :

```
inv([1 1 1 ;2 -3 4;1 1 -1]),
```

On peut aussi le faire en symbolique :

```
inv(sym([1 1 1 ;2 -3 4;1 1 -1])),
inv(sym([1 1 1 ;2 -3 4;1 1 - pi])),
```

On peut calculer des déterminants de matrices : Dans l'exercice 3.4 du TD 3, on veut déterminer l'inversibilité des matrices :

$$\text{a) } A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}, \quad \text{b) } A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & -1 & 1 \\ 2 & 2 & 3 \end{pmatrix}.$$

Il suffit de taper

```
det([1 2;3 1]),
det([1 2 3;
     1 -1 1;
     2 2 3])
```

1.13.4. Équations différentielles

Reprenons l'exercice 7.5 du TD 7 : il fallait résoudre les deux équations différentielles :

$$\begin{aligned}2y'' + 5y' - 3y &= 0, \\2y'' + 5y' - 3y &= t^3 + t^2 - 1.\end{aligned}$$

On écrit :

```
dsolve('2*D2y+5*Dy-3*y=0'),
dsolve('2*D2y+5*Dy-3*y=t^3+t^2-1'),
```

On peut aussi rajouter des conditions aux limites : par exemple, pour traiter la dernière équation différentielle avec

$$y(0) = 1, \quad y'(0) = -8,$$

on écrira

```
dsolve('2*D2y+5*Dy-3*y=t^3+t^2-1', 'y(0)=1', 'Dy(0)=-8'),
```

On peut aussi écrire pour que l'affichage soit plus beau :

```
pretty(dsolve('2*D2y+5*Dy-3*y=t^3+t^2-1', 'y(0)=1', 'Dy(0)=-8')),
```

On pourra aussi traiter grâce à matlab l'équation différentielle du problème 7.1 page 36 du TD 7. Dans un premier temps, on résoud l'équation différentielle :

$$v''(x) + \omega_0^2 v(x) = K \sin(\omega x),$$

en tapant sous matlab

```
pretty(dsolve('D2y+omega0^2*y=K*sin(omega*x)', 'x')),
```

On prend ensuite en compte les conditions initiales

$$v(0) = v(L) = 0,$$

en tapant sous matlab :

```
pretty(dsolve('D2y+omega0^2*y=K*sin(omega*x)', 'y(0)=0', 'y(L)=0', 'x')),
```

On pourra ensuite résoudre l'équation différentielle avec $\omega_0 = \omega$:

$$v''(x) + \omega^2 v(x) = K \sin(\omega x), \tag{1.1}$$

avec les conditions aux limites

$$v(0) = v(L) = 0. \tag{1.2}$$

On pourra aussi s'intéresser à la solution de l'équation différentielle (1.1)-(1.2) avec $L = \pi/\omega$ où K est quelconque puis quand $L = \pi/\omega$ et $K = 0$.

1.13.5. Diagonalisation de matrices

Dans le cours, on a diagonalisé la matrice

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

Il suffit de taper

```
A=[2 0 1;1 1 1;-2 0 1];
disp(A);
pause;
[Q,D]=eigs(A);
disp(diag(D));
pause;
disp(Q);
```

On peut aussi déterminer les valeurs propres et le polynôme caractéristique de façon symbolique.

On pourra consulter l'exemple suivant :

```
A=[2 0 1;1 1 1;-2 0 1];
sp=eig(A);
```

```
sps=eig(sym(A));
disp('valeurs propres (numeriques)');
disp(sp);
disp('valeurs propres (symboliques)');
pretty(sps);
pause;

disp('plus grand module des parties imaginaires (numerique)');
disp(max(abs(imag(sp))));
disp('plus grand module des parties imaginaires (symbolique)');
disp(max(abs(eval(imag(sps)))));
pause;
```

```
P=poly(A);
Ps=poly(sym(A));
disp('polynome caracteristique');
disp(P);
disp('polynome caracteristique (symbolique)');
disp(Ps);
```

Il existe une autre façon de calculer le polynôme caractéristique de A (elle est moins jolie et moins concise sous matlab, mais elle montre l'utilisation des fonction `eye` et `collect`) :

```
A=[2 0 1;1 1 1;-2 0 1];
syms x;
Pss=det(sym(A)-x*eye(3));
```

```
disp(Pss);
disp(collect(Pss));
```

Dans l'exercice 8.1 du TD 8, nous avons montré que la matrice suivante n'était pas diagonale :

$$A = \begin{pmatrix} 8 & 12 & 10 \\ -9 & -22 & -22 \\ 9 & 18 & 17 \end{pmatrix}.$$

On s'en convaincra sous matlab, en tapant, une fois que A est rentrée :

```
[Q,R]=eig(A);
[Qs,Rs]=eig(sym(A));
disp(Q);
disp(Qs);
```

1.13.6. Un dernier exemple issu de la RDM

Prenons un exemple de la RDM (ici, je ne justifie pas les équations, on pourra voir par exemple les notes de cours de MQ41, voir [Bas03]). Ici, on prend un exemple très simple mais on pourra étudier des situations beaucoup plus complexes.

Soit la structure de la figure 1.1.

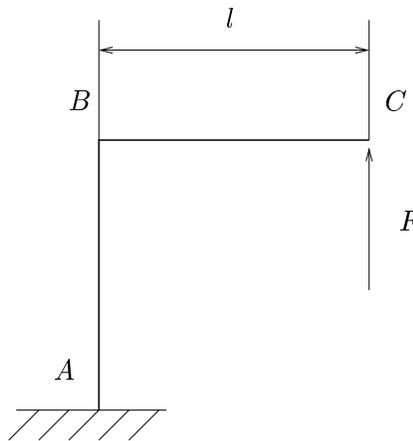


FIG. 1.1 – Un exemple de calcul de structure.

La RDM prévoit que le moment fléchissant dans AB est égal à $M_1(x) = Fl$; il est égal à $M_2(x) = Fx$ dans BC . L'énergie de déformation est égale à

$$W = \frac{1}{2EI} \int_{\text{structure}} M^2 dx = \frac{1}{2EI} \left(\int_0^l M_1^2(x) dx + \int_0^l M_2^2(x) dx \right).$$

Le déplacement vertical du point C est égal à

$$\lambda = \frac{\partial W}{\partial F}.$$

On oublie le terme EI , et sous matlab, on tape

```
syms l F x;  
M1=F*l;  
M2=F*x;  
W=int (M1^2+M2^2,x,0, l );  
diff (W,F)
```

Dérivées et gradients

Dans ce TP, on étudiera quelques applications des dérivées et des gradients, en utilisant les graphiques et la partie symbolique de matlab.

2.1. Initiation au matlab symbolique

On renvoie à la section 1.12 du TP 1.

2.2. Quelques calculs de dérivées symboliques

2.2.1. Calcul d'une dérivée n -ième

On rappelle que l'on a vu en TD la propriété suivante : soit la fonction g de \mathbb{R} dans \mathbb{R} définie par

$$g(x) = (x^3 + 2x - 7) e^x. \quad (2.1)$$

Alors, pour tout entier n , il existe des réels a_n , b_n et c_n tels que

$$g^{(n)}(x) = (x^3 + a_n x^2 + b_n x + c_n) e^x. \quad (2.2)$$

On a montré que ces réels étaient définis par la récurrence :

$$\forall n \in \mathbb{N}, \quad \begin{cases} a_{n+1} = a_n + 3, \\ b_{n+1} = b_n + 2a_n, \\ c_{n+1} = b_n + c_n, \end{cases} \quad (2.3)$$

avec l'initialisation

$$\begin{cases} a_0 = 0, \\ b_0 = 2, \\ c_0 = -7. \end{cases} \quad (2.4)$$

On a aussi donné l'expression explicite de ces coefficients :

$$\forall n \in \mathbb{N}, \quad \begin{cases} a_n = 3n, \\ b_n = 3n^2 - 3n + 2, \\ c_n = n^3 - 3n^2 + 4n - 7. \end{cases} \quad (2.5)$$

QUESTION 2.1. *Écrire une fonction¹ matlab :*

$[a_n, b_n, c_n] = \text{coef_deriv_rec}(n)$

- n est un entier ;

¹Consulter le TP 1, section 1.5.3, page 11, pour les conventions adoptées.

- a_n, b_n et c_n sont définis par (2.2).

Pour cette fonction, on utilisera la récurrence (2.3) et l'initialisation (2.4).

QUESTION 2.2. Écrire une fonction matlab :

$[a_n, b_n, c_n]=\text{coef_deriv_dir}(n)$

- n est un entier ;
- a_n, b_n et c_n sont définis par (2.2).

Pour cette fonction, on utilisera leur expression explicite (2.5).

QUESTION FACULTATIVE 2.3. Écrire une fonction matlab :

$[a_n, b_n, c_n]=\text{coef_deriv_symb}(n)$

- n est un entier ;
- a_n, b_n et c_n sont définis par (2.2).

Pour cette fonction, on utilisera la dérivation symbolique de matlab et la fonction **diff**(f,n) qui permet de dériver n fois une fonction définie à partir d'une variable symbolique.

QUESTION FACULTATIVE 2.4. On pourra comparer les temps de calculs des trois fonctions **coef_deriv_rec**, **coef_deriv_dir** et **coef_deriv_symb**, grâce aux fonctions de matlab **tic** et **toc** (voir en fin de section 1.6 du TP 1 ou l'aide de matlab).

2.2.2. Calcul d'une dérivée en symbolique

On rappelle que l'on a vu en TD la propriété suivante : soient $a, b \in \mathbb{R}_+$ (non simultanément nuls), $x_0 \in \mathbb{R}$, f une fonction de \mathbb{R} dans \mathbb{R} définie au voisinage de x_0 et dérivable en x_0 . Alors

$$\lim_{\substack{h \rightarrow 0 \\ h \neq 0}} \frac{f(x_0 + bh) - f(x_0 - ah)}{(b + a)h} = f'(x_0). \quad (2.6)$$

QUESTION 2.5. Vérifier cette propriété en matlab symbolique : on déclarera a, b, x_0 et h symboliques et on vérifiera (2.6) en utilisant les fonctions **subs**, **limit** et **diff** pour quelques fonctions particulières.

2.3. Étude de fonctions de \mathbb{R}^2 dans \mathbb{R} et tracé des isovaleurs et du gradient

2.3.1. Représentation d'une fonction de \mathbb{R}^2 dans \mathbb{R}

Consulter l'aide de matlab pour comprendre le fonctionnement des fonctions de matlab suivantes :

- **mesh**
- **meshgrid**
- **surf**
- **surfli**
- **contour**

qui permettent de tracer de différentes façon le graphe d'une fonction f de \mathbb{R}^2 dans \mathbb{R} .

On pourra aussi lancer la démonstration de matlab (**demo**) puis consulter les rubriques Graphics, puis 3D plot. On pourra aussi éditer la fonction **peaks** (attention à ne pas la modifier !!) afin de mieux comprendre le fonctionnement des fonctions citées ci-dessus.

QUESTION 2.6. Écrire un script² matlab (commenté) :

trace_3D

• L'utilisateur saisira :

- $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ quatre réels qui définissent le domaine d'étude $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ de f ;
- n_x et n_y le nombre de sous intervalles de $[x_{\min}, x_{\max}]$ et $[y_{\min}, y_{\max}]$;
- c est un entier appartenant à $\{1, 2, 3, 4\}$ avec
 - si $c=1$: tracé avec **mesh** ;
 - si $c=2$: tracé avec **surf** ;
 - si $c=3$: tracé avec **surf** ;
 - si $c=4$: tracé avec **contour** (auquel cas, on rentrera aussi le nombre p d'isovaleurs de f à tracer) ;
- la fonction f , qui sera rentrée sous forme de chaîne de caractères, de façon à pouvoir utiliser ce script pour une fonction :
 - *built-in* ;
 - définie par un fichier *.m ;
 - définie en inline.

On utilisera aussi la fonction **feval**.

- Ce script permet de représenter le graphe d'une fonction de f de \mathbb{R}^2 dans \mathbb{R} , en utilisant l'une des fonctions **mesh**, **surf**, **surf** ou **contour**.

QUESTION 2.7. On testera le script **trace_3D** sur les deux fonctions suivantes :

$$f_1(x, y) = x^2 + 4y^2, \quad (2.7)$$

$$f_2(x, y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - (2x - 10x^3 - 10y^5) e^{-x^2-y^2} - \frac{1}{3} e^{-(x+1)^2-y^2} \quad (2.8)$$

sur le domaine $D = [-3, 3] \times [-3, 3]$ de \mathbb{R}^2 en prenant, par exemple, $n_x = n_y = 30$.

2.3.2. Représentation du gradient d'une fonction de \mathbb{R}^2 dans \mathbb{R}

Consulter l'aide de matlab pour comprendre le fonctionnement de la fonction de matlab **gradient** qui permet de calculer, pour une fonction de \mathbb{R}^2 dans \mathbb{R} son gradient approché.

Consulter de même la fonction **quiver** qui permet de tracer un champ de vecteur (pour une application du plan dans lui même).

On pourra aussi se servir de **demo**.

QUESTION 2.8. Si f est une chaîne de caractère représentant une fonction de \mathbb{R}^2 dans \mathbb{R} (*built-in*, définie par un fichier *.m ou définie en inline) et si x et y sont deux tableaux numériques, constater que la séquence suivante calcule le gradient exact de f en (x, y) :

```
syms vx vy
g=feval(f, vx, vy);
dfx=diff(g, vx);
dfy=diff(g, vy);
xpp=subs(dfx, {vx, vy}, {x, y});
```

²Consulter le TP 1, section 1.5.3, page 11, pour les conventions adoptées.

```

ypp=subs(dfy,{vx,vy},{x,y});
if ~(isnumeric(xpp));
    xpp=eval(xpp);
end
if ~(isnumeric(ypp));
    ypp=eval(ypp);
end

```

QUESTION 2.9. Dédurre de la fonction 2.8, un script :

trace_gradient

- L'utilisateur saisira :
 - $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ quatre réels qui définissent le domaine d'étude $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ de f ;
 - n_x et n_y le nombre de sous intervalles de $[x_{\min}, x_{\max}]$ et $[y_{\min}, y_{\max}]$;
 - la fonction f , qui sera rentrée sous forme de chaîne de caractères, de façon à pouvoir utiliser ce script pour une fonction (built-in, définie par un fichier *.m ou définie en inline).
- Ce script trace les gradient exacts et approchés d'une fonction f de \mathbb{R}^2 dans \mathbb{R} .

QUESTION 2.10. On testera le script **trace_gradient** sur les deux fonctions suivantes (la première a déjà été étudiée en TD) :

$$f_1(x, y) = \sqrt{x^2 + y^2}, \quad (2.9)$$

$$f_2(x, y) = xe^{-x^2-y^2}. \quad (2.10)$$

sur les domaines respectifs $D = [-4, 4] \times [-4, 4]$ (avec $n_x = n_y = 20$) et $D = [-2, 2] \times [-1, 1]$ (avec $n_x = 20$ et $n_y = 15$).

2.3.3. Étude du lien entre les isovaleurs et le gradient d'une fonction \mathbb{R}^2 dans \mathbb{R}

Cette dernière partie vise à vous faire constater de façon empirique sur quelques exemples la propriété vue en cours : le gradient est orthogonal aux isovaleurs et est de norme plus importante là où elles se resserrent.

QUESTION FACULTATIVE 2.11. En modifiant les deux script déjà programmés **trace_3D** et **trace_gradient**, écrire un script

trace_isovaleur_gradient

- L'utilisateur saisira :
 - $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ quatre réels qui définissent le domaine d'étude $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ de f ;
 - n_x et n_y le nombre de sous intervalles de $[x_{\min}, x_{\max}]$ et $[y_{\min}, y_{\max}]$;
 - la fonction f , qui sera rentrée sous forme de chaîne de caractères, de façon à pouvoir utiliser ce script pour une fonction (built-in, définie par un fichier *.m ou en inline) ;
 - p le nombre d'isovaleurs ;
- Ce script permet de tracer en tout point du domaine d'étude le gradient exact de f en ce point. On se servira de la fonction **ginput** qui permet de choisir à la souris un point du domaine d'étude.

QUESTION FACULTATIVE 2.12. *En utilisant le script `trace_ivoaleur_gradient` constater sur les fonctions déjà vues (cf. (2.7), (2.8), (2.9) et (2.10)) la propriété vue en cours, liant le gradient et les isovalues (on prendra garde au fait que les échelles pouvant être différentes en abscisse et ordonnées, le gradient n'apparaîtra pas nécessairement orthogonal aux isovalues).*

On pourra aussi la tester sur l'expression de la fonction (étudiée en TD) définie en polaire par

$$f(r, \theta) = \frac{\cos \theta}{r^2}.$$

Puisque les programmes matlab créés utilisent les coordonnées cartésiennes³, on montrera que l'expression de f en cartésiennes est

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

³On pourrait aussi calculer le gradient exact en polaires. En revanche, le calcul du gradient approché par la fonction **gradient** n'est valable qu'en coordonnées cartésiennes.

Intégrales doubles

Dans ce TP, on utilisera la partie symbolique de matlab pour calculer des intégrales doubles.

On rappelle la propriété suivante vue en cours :

On se donne \mathcal{D} une partie de \mathbb{R}^2 (fermée, bornée et «régulière») et f une application continue de \mathcal{D} dans \mathbb{R} .

On suppose que la partie \mathcal{D} est telle qu'il existe a, b, c, d des réels tels et E et F deux applications telles que

$$\mathcal{D} \subset [a, b] \times [c, d], \quad (3.1)$$

avec, pour tout x appartenant à $[a, b]$, on a

$$(x, y) \in \mathcal{D} \iff y \in [E(x), F(x)]. \quad (3.2)$$

De même, on suppose qu'il existe deux applications G et H telle que, pour tout y appartenant à $[c, d]$,

$$(x, y) \in \mathcal{D} \iff x \in [G(y), H(y)]. \quad (3.3)$$

Pour tout $x \in [a, b]$ fixé, l'application de $[E(x), F(x)]$ dans $\mathbb{R} : y \mapsto f(x, y)$ est continue (donc intégrable) et pour tout $y \in [c, d]$ fixé, l'application de $[G(y), H(y)]$ dans $\mathbb{R} : x \mapsto f(x, y)$ est continue (donc intégrable).

On a alors le théorème :

THÉORÈME 3.1 (Théorème de Fubini). *Sous les notations (3.1), (3.2) et (3.3), on a*

$$\iint_{\mathcal{D}} f(x, y) dx dy = \int_a^b \left(\int_{E(x)}^{F(x)} f(x, y) dy \right) dx = \int_c^d \left(\int_{G(y)}^{H(y)} f(x, y) dx \right) dy. \quad (3.4)$$

QUESTION 3.1. *Écrire un script*

calcul_integrale_double

- *L'utilisateur saisira, avec les notations (3.1), (3.2) et (3.3) :*
 - *la fonction f , qui sera rentrée sous forme de chaîne de caractères, de façon à pouvoir utiliser ce script pour une fonction :*
 - *built-in ;*
 - *définie par un fichier *.m ;*
 - *définie en inline.*
 - *puis*
 - *soit les valeurs a et b et les fonctions E et F ,*

- soit les valeurs c et d et les fonctions G et H .
- Ce script permet de calculer

$$\iint_{\mathcal{D}} f(x, y) dx dy,$$

en utilisant l'une des deux formes données par (3.4).

QUESTION 3.2. Tester le script **calcul_integrale_double** sur les exemples suivants (donnés avec les notations (3.1), (3.2) et (3.3)) :

$$f(x, y) = y^2 \sin(x), \quad a = 0, \quad b = \pi, \quad E(x) = -\sin(x), \quad F(x) = -\sin(x), \quad (3.5)$$

$$f(x, y) = x^2 + y^2, \quad c = -1, \quad d = 1, \quad G(y) = 0, \quad H(y) = (1 - y^2)/2, \quad (3.6)$$

$$f(x, y) = x + y, \quad c = 0, \quad d = 1, \quad G(y) = 0, \quad H(y) = \sqrt{1 - y^2}. \quad (3.7)$$

QUESTION FACULTATIVE 3.3. Pour l'exemple (3.6), peut-on déterminer l'intégrale double de f en utilisant l'autre forme donnée par (3.4) ?

QUESTION FACULTATIVE 3.4. Essayer de trouver des fonctions f pour lesquelles le script **calcul_integrale_double** ne puisse pas déterminer formellement la valeur de l'intégrale double.

Un problème de rectangles

Ce TP facultatif ne fait pas partie du programme de l'UV MT31 ; néanmoins, issu d'un problème rencontré en stage par un étudiant¹ de l'UTBM en stage, il présente un problème de géométrie intéressant et son traitement sous matlab fait appel aux diverses possibilités de matlab : numériques, graphiques, éventuellement symbolique.

4.1. Introduction



FIG. 4.1 – Le problème industriel : consolider des caisses de bois.

On cherche à fabriquer des caisses formées de lattes rectangulaires en bois. Celles-ci sont consolidées par des lattes disposées selon la diagonale d'un rectangle de dimensions connues (voir la figure 4.1). Une des dimensions de ces lattes est connue, l'autre est inconnue.

Géométriquement, on connaît donc un rectangle $ABCD$ et on cherche à construire un rectangle $IJKL$ de telle sorte que :

$$I \in]AD[, \quad J \in]DC[, \quad K \in]CB[, \quad L \in]BA[; \quad (4.1a)$$

$$IJ \text{ est donnée.} \quad (4.1b)$$

Voir figure 4.2 page suivante.

Les résultats théoriques sont préparés en section 4.2 ; ceux-ci pourront être admis. Le TP proprement dit figure en section 4.3.

On pourra faire tourner le fichier matlab `probleme_rectangle_rac.p`, disponible à l'adresse habituelle : <http://utbmjb.chez.tiscali.fr/> Ce fichier est un fichier identique aux habituels fichiers

¹Merci à Pascal Walter de m'avoir fourni les renseignements relatifs à ce problème.

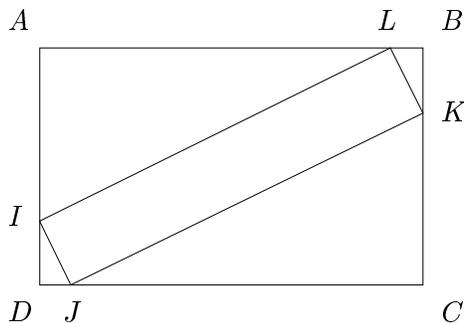


FIG. 4.2 – Le problème géométrique

*.m : il est utilisé comme une fonction *.m mais a été codé² et ne peut être édité (pour que vous travailliez un peu ...). Une aide `aide_probleme_rectangle_rac.txt` est aussi disponible.

Vous pourrez visualiser, grâce à ce fichier, différents cas de figure selon les valeurs des dimensions $a > 0$ et $b > 0$ du «grand» rectangle et la dimension imposée $l > 0$ du «petit» rectangle. On pourra aussi visualiser les deux courbes, dont l'intersection (vide ou non) fournit les solutions au problème (voir section 4.2.2). Dans la section 4.4 page 37, on trouvera quelques courbes, produites par `probleme_rectangle_rac.p`, selon qu'il y ait une solution (voir figures 4.4 et 4.5), trois solutions (voir figure 4.6) ou aucune solution (voir figure 4.7).

4.2. Résultats théoriques

4.2.1. Transformation du problème

On effectue tout d'abord l'analyse du problème (condition nécessaire : voir section 4.2.1.1), puis la synthèse (condition suffisante : voir section 4.2.1.2).

4.2.1.1. Condition nécessaire.

On suppose le problème exposé en section 4.1 déjà résolu (le rectangle $IJKL$ vérifiant (4.1) est construit) et on essaye de caractériser la solution à partir des données.

Montrer que l'intersection des diagonales du rectangle $IJKL$ est aussi l'intersection des diagonales du rectangle $ABCD$. On pourra noter O l'intersection des diagonales du rectangle $ABCD$, considérer s_O la symétrie centrale de centre O et montrer que $I'J'K'L'$, l'image de $IJKL$ par s_O , est égale à $IJKL$.

On est donc ramené au problème suivant : on cherche à construire I et J tels que :

$$I \in]AD[, \quad J \in]DC[; \quad (4.2a)$$

$$OI = OJ, \quad (4.2b)$$

$$IJ = l, \text{ où } l \text{ est donné.} \quad (4.2c)$$

Les deux points K et L sont alors obtenus comme les symétriques respectifs de I et J par rapport à O .

Voir figure 4.3 page suivante.

²grâce à la fonction matlab `pcode`.

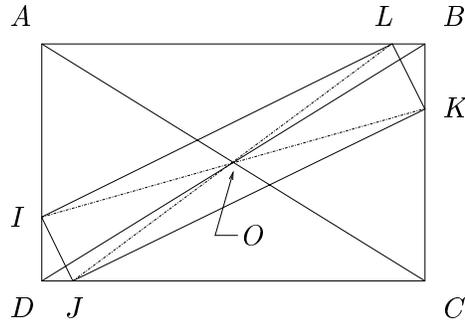


FIG. 4.3 – Le problème géométrique

4.2.1.2. *Condition suffisante.*

Montrer que si l'on construit $I \in]AD[$ et $J \in]DC[$ vérifiant (4.2), puis K et L , les symétriques respectifs de I et J par rapport à O , alors $IJKL$ est un rectangle vérifiant (4.1).

4.2.2. **Problème analytique**

On munit maintenant la figure d'un repère orthonormé direct (D, \vec{I}, \vec{J}) , construit sur $(D, \overrightarrow{DC}, \overrightarrow{DA})$. Dans ce repère, on note $(x, 0)$ les coordonnées de J et $(0, y)$ les coordonnées de I . On pose $a = DC$ et $b = BC$.

1. Montrer que (4.2) est équivalent au système non linéaire de deux équations à deux inconnues :

$$x^2 - y^2 - ax + by = 0 ; \quad (4.3a)$$

$$x^2 + y^2 = l^2, \quad (4.3b)$$

où $(x, y) \in]0, a[\times]0, b[$ et a, b et l sont connus.

2. Cette question peut-être omise.

- (a) Quel est l'ensemble \mathcal{C} des points du plan de coordonnées (x, y) vérifiant (4.3b) ?
 (b) On étudie maintenant \mathcal{H} , l'ensemble du plan des points de coordonnées (x, y) vérifiant (4.3a).

- (i) On se place dans le nouveau repère $(\Omega, \vec{I}, \vec{J})$ où, dans le repère (D, \vec{I}, \vec{J}) , Ω a pour coordonnées $(a/2, b/2)$. Montrer que, dans le repère $(\Omega, \vec{I}, \vec{J})$, \mathcal{H} admet pour équation

$$X^2 - Y^2 = \frac{a^2 - b^2}{4}, \quad (4.4)$$

avec

$$\begin{cases} X = x - \frac{a}{2}, \\ Y = y - \frac{b}{2}. \end{cases} \quad (4.5)$$

- (ii) On change encore de repère et on se place maintenant dans le repère orthonormé direct $(\Omega, \vec{U}, \vec{V})$ où \vec{U} est colinéaire à $\vec{I} + \vec{J}$ et \vec{V} est colinéaire à $\vec{I} - \vec{J}$.

Montrer que, dans le repère $(\Omega, \vec{U}, \vec{V})$, \mathcal{H} admet pour équation

$$X'Y' = \frac{b^2 - a^2}{8}, \quad (4.6)$$

avec

$$\begin{cases} X' = \frac{\sqrt{2}}{2}X + \frac{\sqrt{2}}{2}Y, \\ Y' = -\frac{\sqrt{2}}{2}X + \frac{\sqrt{2}}{2}Y. \end{cases} \quad (4.7)$$

(iii) Quelle est la nature des points de coordonnées (X', Y') vérifiant (4.6) ?

(c) En déduire le tracé de \mathcal{C} et de \mathcal{H} dans le repère (D, \vec{I}, \vec{J}) , selon que $a > b$ ou $a < b$.

(d) Quel est le nombre possible de solutions (x, y) de (4.3) ?

3. On étudie maintenant (4.3) dans le cas où $a = b$.

(a) Montrer que, à x fixé dans $]0, a[$, l'équation (4.3a) admet deux solutions, notées $y_1(x)$ et $y_2(x)$, et définies par

$$y_1(x) = \begin{cases} x & \text{si } x \in]0, a/2], \\ a - x & \text{si } x \in [a/2, a[; \end{cases} \quad (4.8a)$$

et

$$y_2(x) = \begin{cases} a - x & \text{si } x \in]0, a/2], \\ x & \text{si } x \in [a/2, a[. \end{cases} \quad (4.8b)$$

(b) En reportant ces deux valeurs dans (4.3b), et en étudiant, pour $k \in \{1, 2\}$, la fonction $\phi_k : x \mapsto x^2 + y_k^2(x)$, montrer que le système (4.3) vérifie :

• si $l \in]0, \sqrt{2}a/2]$, il existe une unique solution :

$$\begin{cases} x = \frac{\sqrt{2}}{2}l, \\ y = \frac{\sqrt{2}}{2}l. \end{cases} \quad (4.9a)$$

• si $l \in]\sqrt{2}a/2, a]$, il existe trois solutions :

$$\begin{cases} x = \frac{1}{2}(a - \sqrt{2l^2 - a^2}), \\ y = \frac{1}{2}(a + \sqrt{2l^2 - a^2}), \end{cases} \quad \text{ou} \quad \begin{cases} x = \frac{1}{2}(a + \sqrt{2l^2 - a^2}), \\ y = \frac{1}{2}(a - \sqrt{2l^2 - a^2}), \end{cases} \quad \text{ou} \quad \begin{cases} x = \frac{\sqrt{2}}{2}l, \\ y = \frac{\sqrt{2}}{2}l. \end{cases} \quad (4.9b)$$

• si $l \in]a, \sqrt{2}a]$, il existe une unique solution :

$$\begin{cases} x = \frac{\sqrt{2}}{2}l, \\ y = \frac{\sqrt{2}}{2}l. \end{cases} \quad (4.9c)$$

- si $l > \sqrt{2a}$, il n'y a pas de solution.

4. On étudie maintenant (4.3) dans le cas où $a \neq b$; dans ce cas, les solutions sont beaucoup plus difficiles à expliciter³. On montre dans cette question que le système (4.3) peut se ramener à une équation polynômiale de degré 4, que matlab sait résoudre.

(a) Montrer que (4.3) est équivalent à

$$y = \frac{1}{b} (-2x^2 + ax + l^2) ; \quad (4.10a)$$

$$x^2 + \frac{1}{b^2} (-2x^2 + ax + l^2)^2 = l^2. \quad (4.10b)$$

(b) Montrer que (4.10) est équivalent à

$$y = \frac{1}{b} (-2x^2 + ax + l^2) ; \quad (4.11a)$$

$$4x^4 - 4ax^3 + (b^2 + a^2 - 4l^2)x^2 + 2al^2x + l^4 - b^2l^2 = 0, \quad (4.11b)$$

où $(x, y) \in]0, a[\times]0, b[$. Quel est le nombre possible de solutions (x, y) de (4.11) ?

4.2.3. Existence et unicité de la solution

Cette section est facultative.

On verra, dans la partie pratique, que, selon les valeurs des paramètres a , b et l , il existe entre zéro et quatre solutions du problème initial. L'étude du nombre des solutions est difficile dans le cas général ; néanmoins, si on se restreint à la recherche d'une solution «en bas à gauche», la recherche est plus facile. Plus précisément, nous étudions le problème suivant : on cherche à construire un rectangle $IJKL$ de telle sorte que :

$$I \in]DE[, \quad J \in]DF[, \quad K \in]CB[, \quad L \in]BA[;$$

$$IJ \text{ est donnée ;}$$

ici, E désigne le milieu de $[AD]$ et F le milieu de DC . En raisonnant comme dans la section 4.2.2, on montre alors que le nouveau problème est :

$$x^2 - y^2 - ax + by = 0, \quad (4.12a)$$

$$x^2 + y^2 = l^2, \quad (4.12b)$$

où $(x, y) \in]0, a/2[\times]0, b/2[$.

1. Le cas $a = b$ a déjà été traité analytiquement ; en utilisant les résultats (4.9), montrer que le problème (4.12), admet une solution si et seulement si

$$l \in]0, \sqrt{2a}/2[. \quad (4.13)$$

Dans ce cas, la solution est unique.

2. On suppose donc que $a \neq b$.

(a) Selon que $a > b$ ou $a < b$, étudier la fonction g définie par

$$g(x) = 4 \left(x^2 - ax + \frac{b^2}{4} \right).$$

³sur le plan théorique, cela reste possible : voir question 4.4 page 37.

(b) Pour $a > b$, on pose

$$x_1 = \frac{1}{2} \left(a - \sqrt{a^2 - b^2} \right).$$

Grâce aux résultats de la question 2a, étudier les solutions $y \in]0, b/2[$ de (4.12a) pour $x \in]0, a/2[$. On montrera que si $a > b$, pour tout $x \in]0, x_1]$, il existe une unique solution $y(x) \in]0, b/2[$ de (4.12a) définie par

$$y(x) = \frac{1}{2} \left(b - \sqrt{4x^2 - 4ax + b^2} \right), \quad (4.14)$$

et que, si $a < b$, pour tout $x \in]0, a/2[$, il existe une unique solution $y(x) \in]0, b/2[$ de (4.12a) définie par (4.14).

(c) Quel est le sens de variation de la fonction $x \mapsto x^2 + y^2(x)$ sur $]0, x_1[$ ou $]0, a/2[$?

(d) En déduire que le problème (4.12) admet une solution si et seulement si

$$l < \sqrt{\frac{a}{2} \left(a - \sqrt{a^2 - b^2} \right)}, \quad \text{si } a > b, \quad (4.15a)$$

$$l < \sqrt{\frac{b}{2} \left(b - \sqrt{b^2 - a^2} \right)}, \quad \text{si } a < b. \quad (4.15b)$$

De plus, dans ce cas, la solution est unique.

3. En utilisant les résultats (4.13) et (4.15), montrer que, dans tous les cas, le problème (4.12) admet une solution si et seulement si

$$l < \sqrt{\frac{M}{2} \left(M - \sqrt{M^2 - m^2} \right)}, \quad (4.16)$$

où $M = \max(a, b)$ et $m = \min(a, b)$. De plus, dans ce cas, la solution est unique.

4.3. Partie pratique

QUESTION 4.1. En utilisant les résultats (4.9) et (4.11), ainsi que la fonction⁴`roots`, écrire une fonction matlab

`[x, y, v]=probleme_rectangle(a, b, l)`

- a, b et l sont trois réels strictement positifs ;
- x et y sont les éventuelles solutions du problème (4.3) ; v est la maximum des valeurs absolues (s'il y a une solution) de $x^2 + y^2 - l^2$ et de $x^2 - y^2 - ax + by$.

QUESTION 4.2. Améliorer la fonction `probleme_rectangle` pour qu'elle affiche aussi les rectangles obtenus (de façon analogue aux figures 4.4b à 4.7b).

QUESTION FACULTATIVE 4.3. En utilisant les résultats de la question 2 de la section 4.2.2, améliorer la fonction `probleme_rectangle` pour qu'elle affiche aussi les coniques définies par (4.3) (de façon analogue aux figures 4.4a à 4.7a).

⁴Voir section 1.9 page 14 du TP 1.

Pour tracer l'hyperbole d'équation $X^2 - Y^2 = R$, on pourra utiliser le paramétrage trigonométrique hyperbolique suivant

$$\begin{aligned}x &= k \cosh(\theta), \\x &= k \sinh(\theta).\end{aligned}$$

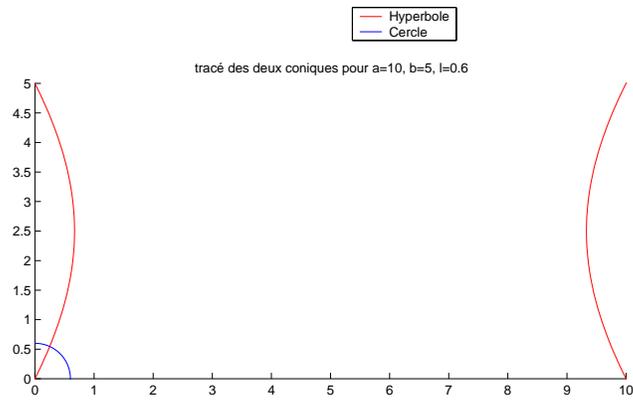
QUESTION FACULTATIVE 4.4. *Sous matlab formel, montrer que les solutions de (4.3) peuvent être déterminées explicitement. On pourra taper*

```
S=solve('x^2+y^2=l^2','x^2-y^2-ax+by=0','x,y');
pretty(S.x);
pretty(S.y);
```

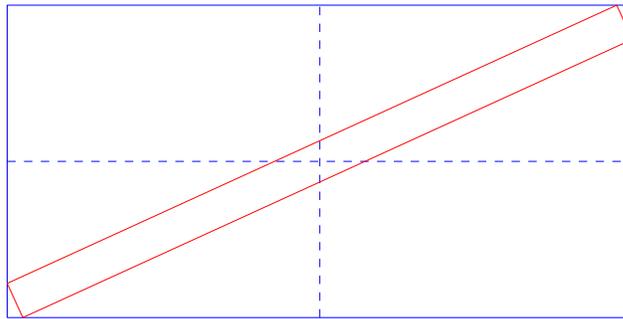
QUESTION FACULTATIVE 4.5. *De même, montrer que les solutions de (4.11) peuvent être déterminées explicitement.*

4.4. Exemples de simulations numériques

Dans cette section se trouvent quelques courbes produites par `probleme_rectangle_rac.p`.

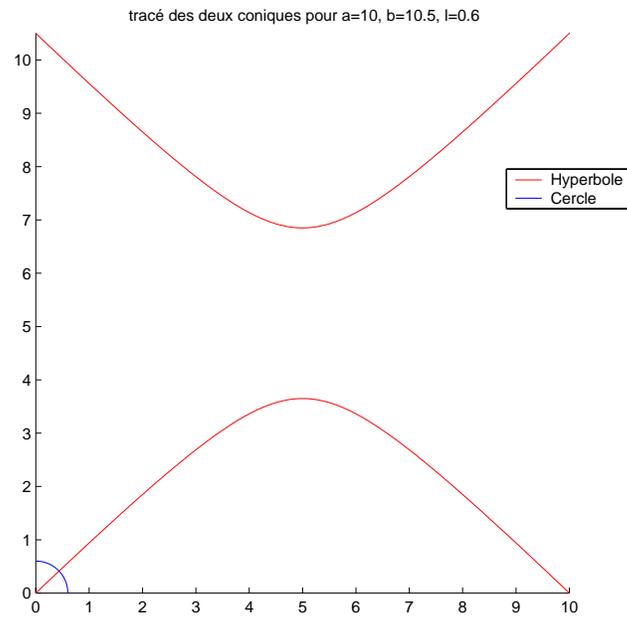


(a)

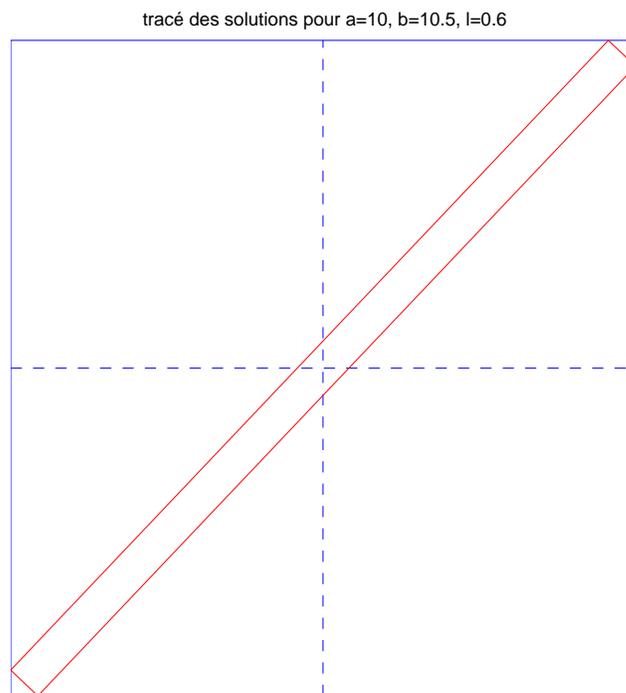
tracé des solutions pour $a=10$, $b=5$, $l=0.6$ 

(b)

FIG. 4.4 – Les solutions pour $a = 10$, $b = 5$ et $l = 0,6$.

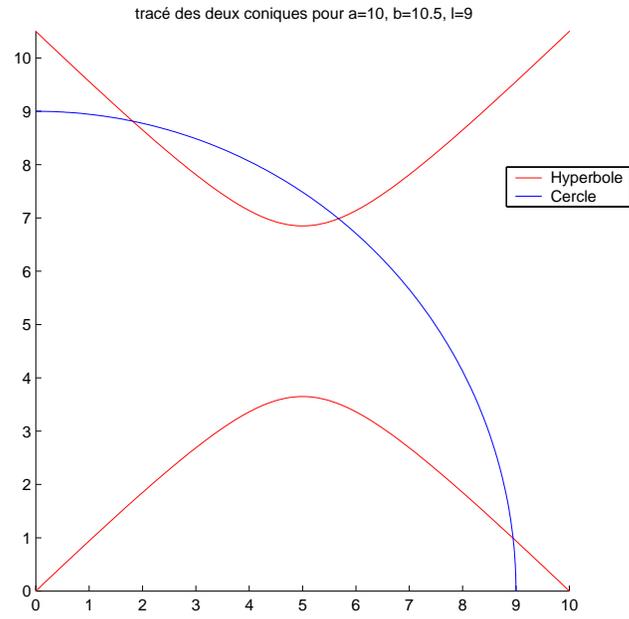


(a)

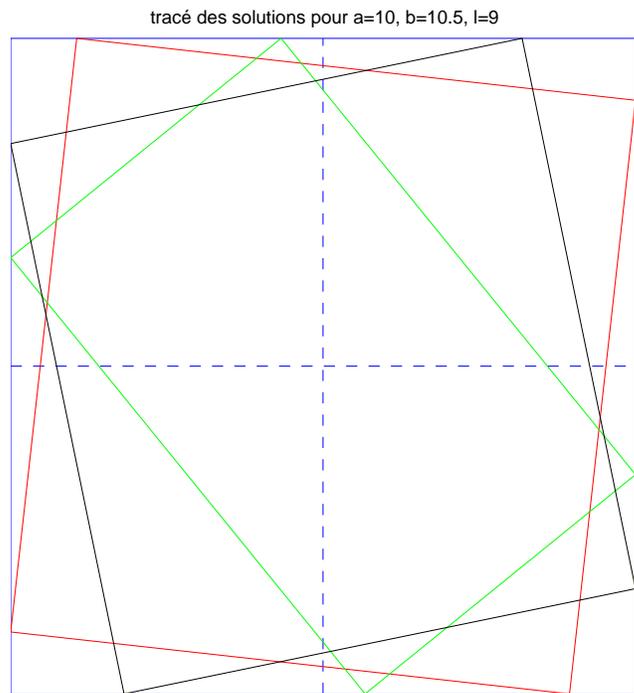


(b)

FIG. 4.5 – Les solutions pour $a = 10$, $b = 10, 5$ et $l = 0, 6$.

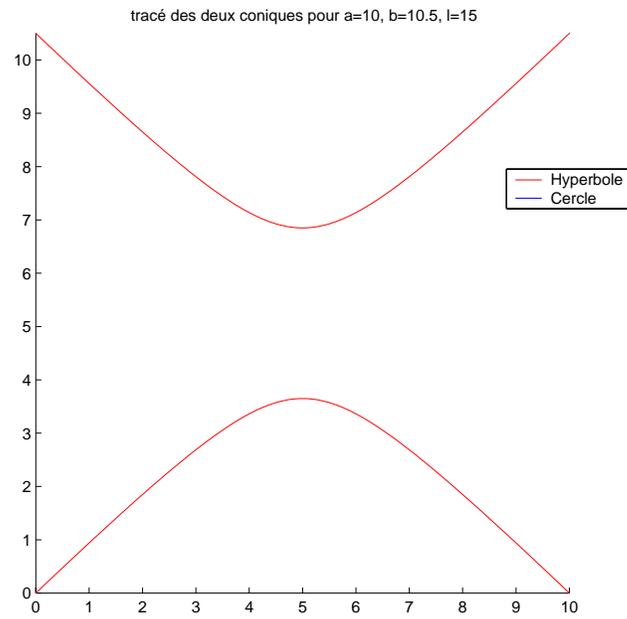


(a)

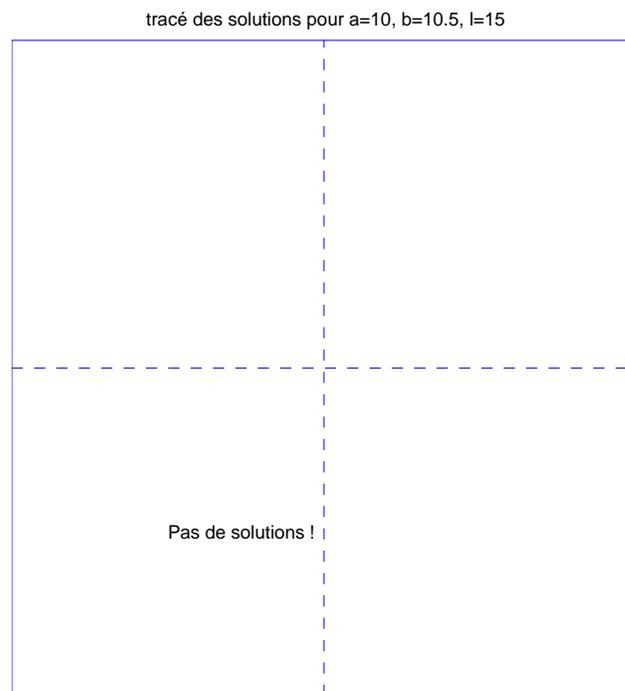


(b)

FIG. 4.6 – Les solutions pour $a = 10, b = 10,5$ et $l = 9$.



(a)



(b)

FIG. 4.7 – Les solutions pour $a = 10$, $b = 10,5$ et $l = 15$.

Bibliographie

- [Bas03] Jérôme Bastien. Résistance des matériaux, Introduction aux calculs des structures. Notes de cours de MQ41 de l'UTBM, disponible sur le web : <http://utbmjb.chez.tiscali.fr/>, rubrique MQ41, 2003.
- [BM03] Jérôme Bastien et Jean-Noël Martin. *INTRODUCTION À L'ANALYSE NUMÉRIQUE. Applications sous matlab*. Dunod, 2003.
- [HLR01] Brian R. Hunt, Ronald L. Lipsman et Jonathan M. Rosenberg. *A guide to matlab, for beginners and experienced users*. Cambridge University Press, 2001.
- [MM97] Mohand Mokhtari et Abdelhalim Mesbah. *Apprendre à maîtriser matlab*. Springer-Verlag, 1997.
- [Mok00] Mohand Mokhtari. *Matlab 5.2 & 5.3 et Simulink 2 & 3 pour étudiants et ingénieurs*. Springer-Verlag, 2000.
- [PES99] Eva Pärt-Enander et Anders Sjöberg. *The matlab 5 handbook*. Addison-Wesley, 1999.
- [Ref92] *Matlab, Reference guide*. The Math Work, Inc., 1992.