Deuxième année
FICM

Département Énergie & Fluides

Utilisation de logiciels scientifiques

Introduction à Mathematica

- TP -

version 1.2.1

Mathieu Jenny
Vincent Schick

Année universitaire 2017 – 2018

Table des matières

Introduction générale

Mines Nancy bénéficie depuis plusieurs années d'une license globale de type « campus » du ***logiciel de calcul formel et numérique*** **Mathematica**. Le but de ces TP est de vous initier à ce logiciel, qui sera utilisé dans ***les modules de mécanique des fluides et de méthodes numériques*** de cette année. Vous pourrez aussi l'utiliser lors de projets, etc... durant toute votre scolarité. De plus, ***il est important pour un ingénieur de connaître au moins un logiciel scientifique*** car ceux-ci sont largement utilisés en entreprises pour la R&D. Il existe bien entendu d'autres logiciels de ce type que Mathematica. Nous ne citerons ici que Matlab qui est l'un des plus largement utilisé.

Les séances de TP seront encadrés par Mathieu Jenny et Vincent Schick. Le planning des TP est le suivant :

Date	Horaire	Salle
mercredi 13 septembre	13h30 – 17h30	P318
jeudi 14 septembre	13h30 – 16h30	P318
mercredi 20 septembre (TP noté)	13h30 – 16h30	P318

Notez bien que ***le troisième TP sera noté*** et comptera pour l'évaluation du module. La note de TP sera basée sur votre attitude pendant le TP (autonomie) et sur un compte-rendu, que vous transmettrez sur Arche `https://arche.univ-lorraine.fr/` à la fin de la séance. Le compte-rendu comportera vos réponses aux questions de l'énoncé ainsi que vos programmes ou notebooks Mathematica.

En tant qu'étudiant à Mines Nancy, vous pouvez installer gratuitement la dernière version de Mathematica sur votre machine dans le cadre de la licence site de l'établissement.

Pour cela, il faut :

— créer un compte sur `user.wolfram.com` avec votre adresse mail de l'Université de Lorraine,

— puis, si vous êtes étudiant, se connecter à cette adresse
`https://user.wolfram.com/portal/requestAK/3c21ddfefe2c197d7aa8895226fbf78bd9f1971c`
pour obtenir une clef d'activation.

Après inscription, vous recevrez un mail vous indiquant un lien de téléchargement. Vous pourrez également télécharger Mathematica depuis votre compte sur `user.wolfram.com`, dans l'onglet "my products and services".

TP 1

Initiation

Le sujet a été adapté d'un sujet de TP original de Gérard Vinsard, enseignant-chercheur de l'ENSEM.

1.1 Introduction

L'objectif de ce TP est de s'initier à Mathematica en manipulant différentes commandes de façon ***expérimentale***. Toutes les commandes Mathematica apparaissent en **gras soulignés**.

Un petit 'truc' d'intérêt général : quand Mathematica « pédale », c'est peut-être dû au fait qu'il a gardé en mémoire toute l'histoire de vos erreurs passées. Pour faire table rase il est parfois très utile de faire un `Quit Kernel`, commande accessible dans l'onglet 'Evaluation' du menu principal.

1.2 Initiation : de l'instruction à sa signification

Pour chaque section, créer un nouveau programme (fichier *.wl ou *.wls) et le sauvegarder. Les packages *.wl doivent être lancés sous Mathematica alors que les scripts *.wls sont autonomes et peuvent être lancés en ligne de commandes. Il existe un troisième format de programmation sous Mathematica qui est un simple fichier texte (*.txt). Ce dernier peut être exécuté dans un notebook (*.nb) avec la commande

```
1  <<monprg.txt
```

Le notebook ne doit pas être utilisé pour programmer. Sa vocation est d'être une interface où l'on lance directement les commandes les unes à la suite des autres. Ces commandes peuvent être l'exécution d'un programme ou script (*.wl, *.wls ou *.txt). Un notebook peut aussi servir à la mise au point d'un script.

Lorsque l'on exécute un script dans un notebook, il faut que Mathematica puisse trouver le fichier du script. Pour cela, Mathematica cherche le fichier dans le répertoire de travail. Pour savoir dans quel répertoire de travail on est, il faut taper

```
2  Directory[]
```

Pour modifier le répertoire de travail, on utilise la commande

```
3  SetDirectory[]
```

Voyez aussi sur ce sujet la section 1.2.7.

1.2.1 Recherche d'aide

L'utilisation de l'aide en ligne est fondamentale pour pouvoir utiliser Mathematica. Grâce à cette aide, on peut retrouver la syntaxe des différentes fonctions et commandes de Mathematica. Vous serez souvent amenés à l'utiliser pour retrouver une commande ou sa syntaxe.

```
4  ?Print
```

```
5  Information[Print]
```

On peut aussi taper une commande, la surligner avec la souris puis taper `F1` pour obtenir une fenêtre d'aide.

1.2.2 Définition de variables

Mathematica, comme la plupart des environnements de programmation avancé, permet de définir et de manipuler des variables, au sens informatique du terme, en cachant à l'utilisateur la définition de leur type (réel, entier, chaîne de caractères, *etc* ...). Cette définition est indispensable pour interpréter le code binaire du contenu de la variable. Par exemple, le code binaire du caractère « 0 »en ASCII est 00110000. La même information binaire interprétée comme un entier codé en 8 bits donne $0*2^7+0*2^6+1*2^5+1*2^4+0*2^3+0*2^2+0*2^1+0*2^0=48$ ou encore pour des entiers codés sur 4 bits un tableau de deux entiers : 3 et 0.

On comprend donc que même si Mathematica simplifie la manipulation de variables, il faut garder à l'esprit que l'on ne peut pas mélanger tous les types de donnée sans précaution. De plus, Mathematica va plus loin que la simple définition du type de donnée : le contenu d'une variable est pour lui un objet, c'est-à-dire une entité mathématique (une matrice par exemple) comportant des données (des nombres) et un ensemble de règles opératoires (addition, multiplication, etc ...). C'est d'ailleurs ce qui fait la valeur ajoutée du logiciel.

Nous allons illustrer cela avec quelques exemples de commandes.

```
6  x=1;Print["x= ",x];x=.;Print["x= ",x];
```

```
7  x=1;Print["x= ",x];Clear[x];Print["x= ",x];
```

```
8  x=1;y=3;Print["x+y= ",x+y];Clear[x];Print["x+y= ",x+y];x={1,2};Print["x+y= ",x+y];
```

```
9  x=1;expr=x^2+1;Print["expr= ",expr];x=.;Print["expr= ",expr];
```

```
10 x=1;expr:=x^2+1;Print["expr= ",expr];x=.;Print["expr= ",expr];
```

1.2.3 Variables locales

```
11 x=1;Block[{x=2},Print["x= ",x]];Print["x= ",x]
```

```
12 Clear[s,i];
13 Block[{s=0,i},For[i=1,i<10,i=i+1,s=s+i];s]
14 Print[s," ",i]
```

1.2.4 Calcul symbolique

Mathematica a d'abord été pensé pour faire du calcul algébrique. Il permet donc de manipuler des expressions symboliques et non pas seulement numériques. Nous allons voir quelques exemples afin d'avoir un aperçu des capacités de Mathematica et afin de dégager quelques règles d'utilisation.

Simplifications

```
15 Clear[x,a,b,c];y=Expand[(x-a)*(x-b)*(x-c)];Factor[y]
```

```
16 Clear[x];Print["Simplify[Sqrt[x^2]]= ",Simplify[Sqrt[x^2]]];
17 Print["Assuming[{x>0},Simplify[Sqrt[x^2]]]= ",Assuming[{x>0},Simplify[Sqrt[x^2]]]];
```

```
Clear[x]; Print["Simplify[Sqrt[x^2]]= ", Simplify[Sqrt[x^2]]];
Print["Assuming[{x<0}, Simplify[Sqrt[x^2]]]= ", Assuming[{x<0}, Simplify[Sqrt[x^2]]]];
```

```
Clear[x]; Simplify[Cos[2*x]-2*Cos[x]^2+1]
```

Définition de fonctions

```
f[x_]:=x^2; Print[f[y]]; Plot[f[x], {x, -1, 1}]
```

```
f=Function[{x}, x^2]; Print[f[y]]; Plot[f[x], {x, -1, 1}]
```

```
Print[Function[{x}, x^2][y]]; Plot[Function[{x}, x^2][y], {y, -1, 1}]
```

```
f=Function[{x}, If[x<0, -x, x]]; Plot[f[y], {y, -1, 1}]
```

```
f=Function[{x}, Piecewise[{{-x, x<0}}, x]]; Plot[f[y], {y, -1, 1}]
```

Une fonction utile pour l'affichage

```
print[strx_]:=Print[strx, " = ", ToExpression[strx]]
x=1; print["x"]
x=y^2; print["x"]
```

Équations algébriques

```
Clear[x]; Print["x /. x->1= ", x /. x->1];
Print["x /. {x->1}= ", x /. {x->1}];
Clear[y]; Print["x+y /. {x->1, y->2}= ", x+y /. {x->1, y->2}]
Print["x+y /. {x->y, y->1}= ", x+y /. {x->y, y->1}]
```

```
Clear[x, y]; sol=Solve[{2*x+y==1, x+2*y==1}, {x, y}];
{x, y} /. sol[[1]]
```

```
Clear[x]; sol=Solve[x^2-1==0, x]; Print[sol];
x /. sol[[1]]
```

Équations différentielles ordinaires

```
Print["D[x^2, x]= ", D[x^2, x]]; Print["D[x^2, {x, 2}]= ", D[x^2, {x, 2}]];
Print["D[x^2*y^2, {x, 1}, {y, 2}]= ", D[x^2*y^2, {x, 1}, {y, 2}]]
```

```
Clear[x, y]; sol=DSolve[{D[y[x], x]+y[x]==0, y[0]==1}, y, x];
y=Evaluate[y /. sol[[1]]]; Print["y= ", y]; Print["y[t]= ", y[t]]
```

```
Clear[x, y];
sol=DSolve[{D[y[x], {x, 2}]+y[x]==0, y[0]==1, Evaluate[D[y[x], x] /. {x->0}]==0}, y, x];
y=Evaluate[y /. sol[[1]]]; Print["y= ", y]; Print["y[t]= ", y[t]]
```

```
Clear[x, y]; sol=DSolve[{y''[x]+y[x]==0, y[0]==1, y'[0]==0}, y, x];
y=Evaluate[y /. sol[[1]]]; Print["y= ", y]; Print["y[t]= ", y[t]]
```

Intégrales

```
Clear[x];Print["Integrate[1/(1+x^2),x]= ",Integrate[1/(1+x^2),x]];
Print["Integrate[1/(1+x^2),{x,0,1}]= ",Integrate[1/(1+x^2),{x,0,1}]];
```

1.2.5 Programmation

Nous avons vu jusqu'à présent comment utiliser Mathematica en mode « notebook » avec des instructions simples. Cependant, afin d'effectuer des tâches complexes et de les automatiser, il faut disposer d'un langage de programmation permettant de définir des conditions d'exécution et des boucles.

Conditions

```
Print["x=.; If[x<0,-1,1,\"on ne sait pas\"]= ",x=.;If[x<0,-1,1,"on ne sait pas"]]
Print["x=-1; If[x<0,-1,1,\"on ne sait pas\"]= ",x=-1;If[x<0,-1,1,"on ne sait pas"]]
```

```
Clear[x];Plot[Piecewise[{{0,x<-1},{x+1,x<0},{1-x,x<1}},0],{x,-2,2}]
```

Boucles

```
s=0;For[i=1,i<10,i=i+1,s=s+i];Print["s= ",s]
```

```
s=0;i=1;While[i<10,s=s+i;i=i+1];Print["s= ",s]
```

```
s=0;i=0;While[i<9,i=i+1;s=s+i];Print["s= ",s]
```

```
n=10;x=Table[0,{i,n}];y=Table[0,{i,n}];
For[i=1,i<n+1,i++,x[[i]]=2 Pi (i-1)/(n-1);y[[i]]=Sin[2 Pi (i-1)/(n-1)];];
xy=Transpose[{x,y}];g=ListLinePlot[xy,PlotMarkers -> {"+",Large}];
Print[Show[g,PlotRange -> {{0,2 Pi},{-1,1}},AxesOrigin -> {0,0}]];
```

On pourra comparer cette dernière série de commandes avec :

```
f[x_]:=Sin[x];Plot[f[x],{x,0,2 Pi}]
```

1.2.6 Calcul numérique

Ce que nous entendons par calcul numérique, c'est la manipulation de nombres et de tableaux de nombres comme le ferait une simple calculatrice. Cela doit être différencié du calcul symbolique où les expressions algébriques sont directement manipulées. Ici, une fonction n'est connue que par ses valeurs numériques en un nombre fini de points discrets qui sont stockés dans un tableau.

Tableaux et/ou liste

```
x=Array[Function[i,i^2],10];Print["x =",x];Print["x[[3]] = ",x[[3]]]
```

```
A=Array[Function[{i,j},i^2+j],{10,10}];Print["A= ",A];Print["A[[6,2]]= ",A[[6,2]]];
Print["A[[6]]= ",A[[6]]];Print["A[[6]][[2]]= ",A[[6]][[2]]];
```

```
x={};Print["x= ",x];For[i=0,i<=10,i=i+1,x=Join[x,{i^2}]];Print["x= ",x]
```

```
x={1,2,3,4};Print["Rest[x]= ",Rest[x]," ","First[x]= ",First[x]]
```

```
n=5;x=Table[0,{i,n},{j,n}];x[[1;;n,2]]=1;Print[MatrixForm[x]];
```

Algèbre linéaire

```
x=Array[Function[i,i],3];y=Array[Function[i,-i],3];
Print["x+y=",x+y];Print["x*y=",x*y];Print["x.y=",x.y];
```

```
A=Array[Function[{i,j},i+j-1],{3,3}];x=Array[Function[i,i],3];
Print["A.x=",A.x];Print["x.A=",x.A];
Print["Dot[A,x]=",Dot[A,x]];Print["Dot[x,A]=",Dot[x,A]];
```

```
A=Array[Function[{i,j},i^(j-0)],{3,3}];
Print["Inverse[A]=",Inverse[A]];
Print["LinearSolve[A,A.{1,1,1}]=",LinearSolve[A,A.{1,1,1}]];
```

1.2.7 Entrée / Sortie

Sorties graphiques

Nous l'avons vu précédemment, Mathematica permet de tracer des graphiques. On peut aussi tracer des lignes de niveau :

```
Clear[x,y];ContourPlot[x^2+y^2,{x,-1,1},{y,-1,1}]
ContourPlot[x^2+y^2,{x,-1,1},{y,-1,1},ContourShading->None]
```

Manipulation de fichiers

Enfin, un programme, le contenu d'une variable, *etc* ... peut être sauvegardé dans des fichiers. Nous donnons quelques commandes pour se déplacer dans l'arborescence des répertoires, pour créer et lire des fichiers de sauvegarde.

```
Print["Directory[]=",Directory[]];
Print["FileNames[]=",FileNames[]];
Clear[x];Put[Cos[x],"sauvegarde.m"];
Print["FileNames[]=",FileNames[]];
Print["Get[\"sauvegarde.m\"]=",Get["sauvegarde.m"]];
If[StringTake[$System,5]=="Linux",Run["rm sauvegarde.m"],Run["del sauvegarde.m"]]
Print["FileNames[]=",FileNames[]];
```

Quand on écrit un programme pour Mathematica, il est vivement conseillé d'écrire le texte du programme avec un logiciel de traitement de texte ou avec l'éditeur de Mathematica. Créez un fichier texte contenant « Print[“Hello World”];» et sauvegardez-le en l'appellant *monprg.txt* dans votre répertoire courant de travail. Sous Mathematica tapez la commande :

```
<<monprg.txt
```

On vous suggère d'écrire en tête de votre fichier *monprg.txt*, en commentaire, c'est-à-dire entre `(*` et `*)`, les commandes qui permettent de lancer votre programme par copier-coller dans un Notebook.

```
(*
SetDirectory["U:\Mines\Mathematica"];
<<monprg.txt
*)
```

L'avantage de cette façon de procéder est qu'un programme de simulation ou de traitement de données complexe pourra être découpé en sous-programmes (ou 'scripts') de petite taille, bien ciblés, qui seront appelés par le programme principal, qui pourrait avoir la forme suivante (pour un programme de simulation) :

```
(*
SetDirectory["U:\Mines\Mathematica"];
<<mainprog.txt
*)
<<generalcommands.txt
<<parameters.txt
<<initialconditions.txt
<<linearterms.txt
<<nonlinearterms.txt
<<code.txt
```

1.3 Réalisation de tâches

1. Chercher la racine réelle de $\mathbf{x^3 + x = 1}$.
2. Résoudre le système d'équations linéaires :

$$\begin{pmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \frac{\pi^2}{9} \begin{pmatrix} sin(\pi/3) \\ sin(2\pi/3) \\ sin(\pi) \\ sin(4\pi/3) \\ sin(5\pi/3) \end{pmatrix}$$

3. Résoudre : $\begin{cases} \dfrac{dx}{dt} + x &= \cos t \\ x(0) &= x(2\pi) \end{cases}$
4. Calculons la distance moyenne quadratique entre un point et un disque. Si le point a pour coordonnées (u, v) et le disque est composé des points $D = \{(x, y)$ tel que $x^2 + y^2 < a^2\}$, la distance moyenne est

$$\sqrt{\frac{1}{\pi a^2} \int_{(x,y)\in D} ((x-u)^2 + (y-v)^2)\, dxdy}.$$

   La difficulté est d'exprimer cette intégrale sans jamais effectuer un calcul soi même.

TP 2

Utilisation en mathématiques et physique

Le sujet a été adapté d'un sujet de TP original d'Emmanuel Plaut, professeur à Mines Nancy.

2.1 Introduction

Dans ce TP, nous allons voir comment utiliser Mathematica pour résoudre des problèmes de physique et de mécanique des fluides.

Dans le cours de méthodes numériques, nous verrons comment transformer une équation aux dérivées partielles en système linéaire. La première partie sur le conditionnement de systèmes linéaires pourrait donc constituer une étape de la résolution numérique d'un problème en mécanique et énergétique.

Enfin, nous allons résoudre un problème d'hydrothermique en utilisant les capacités en calcul formel de Mathematica.

2.2 Conditionnement de systèmes linéaires : influence d'une perturbation du second membre

1. Définissez la matrice

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

Note : pour visualiser la matrice A sous une forme de tableau, afin de détecter d'éventuelles fautes de frappes, vous pouvez utiliser la commande

```
1 TableForm[A]
```

ou

```
2 MatrixForm[A]
```

2. Définissez le vecteur

$$b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}.$$

3. Calculez la solution unique x du système

$$A \cdot x = b$$

de deux manières différentes, en utilisant soit la commande `LinearSolve`, soit la commande `Inverse`.

Le résultat est un vecteur x d'une forme extrêmement simple, c'est-à-dire avec 4 composantes identiques qui prennent la valeur la plus simple que l'on puisse imaginer.

4. Calculez la solution x' d'un système où l'on a légèrement perturbé le second membre,

$$A \cdot x' = b' = b + db = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}.$$

5. Calculez

$$dx = x' - x$$

et constatez que, pour la norme infinie

$$\left\| \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} \right\|_\infty = \max\{|v_i|\},$$

on a une amplification des erreurs relatives :

$$\frac{\|dx\|}{\|x\|} \gg \frac{\|db\|}{\|b\|}. \tag{2.1}$$

On vous demande pour cela de définir une fonction norme infinie

```
3  Normei[v_]:=...
```

de calculer explicitement les deux erreurs relatives quotients apparaissant dans 2.1 :

```
4  errx=Normei[dx]/Normei[x]
5  errb=Normei[db]/Normei[b]
```

et enfin de donner la valeur du coefficient d'amplification de l'erreur `errx/errb`.

2.3 Conditionnement de systèmes linéaires bis :

2.3.1 influence d'une perturbation de la matrice

1. Reprenez le notebook précédent et sauvez les valeurs de A, x et b dans un fichier à l'aide de la commande*

```
6  Save["DefAxb.txt",{A,x,b}]
```

2. Fermez Mathematica, relancez-le à nouveau et ouvrez un nouveau notebook. Lisez les valeurs de A, x et b avec la commande

```
7  Get["DefAxb.txt"]
```

3. Pour être sûr, vérifiez que le booléen

```
8  A.x==b
```

est bien vrai.

*. C'est ici que le `SetDirectory` de la page 1 révèle pour la première fois son utilité.

4. Définissez la matrice légèrement perturbée

$$A' = \begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix}.$$

5. Calculez

$$dA = A' - A$$

et sa norme infinie c'est-à-dire le maximum des valeurs absolues de ses coefficients.

6. Résolvez le système ainsi perturbé

$$A' \cdot x' = b,$$

et calculez

$$dx = x' - x$$

ainsi que sa norme infinie.
On vous demande d'utiliser la même fonction `Normei` pour calculer $\|dA\|_\infty$ et $\|dx\|_\infty$, quitte à revoir votre façon de la définir† de sorte qu'elle soit bien polymorphe (*i.e.* pouvant s'appliquer à des arguments de structures différentes, en l'occurrence des listes à 1 ou 2 niveaux).

7. Montrez qu'ici aussi on a amplification des erreurs relatives :

$$\frac{\|dx\|}{\|x\|} \gg \frac{\|dA\|}{\|A\|}, \tag{2.2}$$

en calculant explicitement ces deux erreurs relatives

```
9  errx=Normei[dx]/Normei[x]
10 errA=Normei[dA]/Normei[A]
```

et donnez enfin la valeur du coefficient d'amplification de l'erreur `errx/errA`.

Commentaires :

On dit pour exprimer ces phénomènes que la matrice A est mal conditionnée. Une étude mathématique plus complète de ce phénomène a normalement été faite lors du cours d'analyse numérique de Xavier Antoine au second semestre de votre première année (voir aussi Ciarlet P. G. Introduction à l'analyse numérique matricielle et à l'optimisation, Masson 1990, dont ces exercices sont tirés).

2.4 Étude d'un problème d'hydrothermique : modèle de vortex polaire

On s'intéresse au vortex polaire que l'on modélise, au voisinage du pôle nord, par un champ de vitesse

$$\vec{v} = v_r(r,z)\vec{e}_r + v_\theta(r,z)\vec{e}_\theta + v_z(r,z)\vec{e}_z \tag{2.3}$$

avec

$$v_r(r,z) = \frac{1}{2}sr, \quad v_\theta(r,z) = \omega r, \quad v_z(r,z) = -sz$$

en coordonnées cylindriques (r,θ,z), r étant la distance à l'axe des pôles, θ la longitude et z l'altitude.

1. Exprimez $\vec{v}$ en coordonnées cartésiennes (x,y,z), en utilisant par exemple les règles de définition de la base locale des coordonnées cylindriques

$$\vec{e}_r = \cos\theta\vec{e}_x + \sin\theta\vec{e}_y, \vec{e}_\theta = \cos\theta\vec{e}_y - \sin\theta\vec{e}_x,$$

†. Par rapport à l'exercice 1.

et une règle de transformation du type

```
regle={Cos[theta]->x/r,Sin[theta]->y/r}
```

2. Vérifiez alors que l'écoulement d'air associé est bien incompressible, *i.e.*

$$\mathrm{div}(\vec{v}) = \partial_x v_x + \partial_y v_y + \partial_z v_z = 0.$$

Retrouvez ce résultat directement en cylindriques, en utilisant la fonction `Div`.

3. Représentez des projections du champ de vitesse $\vec{v}$ dans les plans $(x, 0, z)$ et $(x, y, 10\text{ km})$, à l'aide de la commande `VectorPlot`, suivant par exemple la syntaxe

```
y=0;
gr=VectorPlot[{Vx,Vz},{x,0,10},{z,0,10}]
...
```

Pour cette application numérique vous considérez un vortex polaire donné, dans le système d'unités international, par

$$s = 0.001\text{ Hz}, \quad \omega = 0.05\text{ Hz}\ . \tag{2.4}$$

Vous prendrez soin d'indiquer sur vos graphes les axes de variables et le nom des variables grâce à la commande `Show` et à ses options `Frame` et `FrameLabel` :

```
Show[gr,Frame->True,FrameLabel->{"a","b","c",""}]
```

4. Dans ce qui suit on revient au cas d'un vortex polaire général. Vous pouvez pour cela utiliser un nouveau notebook, ou la commande

```
Clear[s,omega]
```

dans le notebook de cet exercice.

5. On considère une particule d'ozone qui se trouve à $t = 0$ en $(r, \theta, z) = (r_0, 0, h)$. On suppose qu'elle est advectée passivement par l'écoulement, *i.e.* que sa trajectoire dans le vortex polaire est donnée par

$$\begin{cases} \frac{dr}{dt} &= v_r(r(t), z(t)) \\ r(t)\frac{d\theta}{dt} &= v_\theta(r(t), z(t)) \\ \frac{dz}{dt} &= v_z(r(t), z(t)) \end{cases}$$

Calculez, à l'aide de la commande `DSolve`, la trajectoire de la particule $(r(t), \theta(t), z(t))$ dans le cas général.

6. Représentez à l'aide de la commande `ParametricPlot3D` la trajectoire d'une particule d'ozone dans le vortex polaire donné par 2.4, qui serait partie à $t = 0$ de $r_0 = 1$ km, $z_0 = 15$ km, et ce pendant la première heure de « vol »de la particule. À quelle altitude se trouve la particule au bout de cette heure de « vol » ?
Indication : Vous aurez intérêt à augmenter le nombre de points utilisés par `ParametricPlot3D` à l'aide de l'option `PlotPoints` correspondante.

7. On modélise le champ de température au voisinage du pôle nord par une loi de la forme

$$T(r, z) = (a + br^2)(1 - cz).$$

Calculez, en utilisant la commande `Solve`, les valeurs de a, b et c permettant d'assurer les conditions

$$\begin{cases} T(0,0) &= -80°C \\ T(2\text{ km}, 0) &= -79°C \\ T(0, 10\text{ km}) &= -100°C \end{cases}$$

8. En faisant confiance à ce modèle de température, représentez la température de la particule d'ozone au cours de la trajectoire calculée en 4.3, et au cours du temps. Au bout de combien de temps la température de cette particule atteindra t'elle $-70°C$?

   Indication : comme l'équation à résoudre est non algébrique, on fera une résolution numérique approchée à l'aide de la commande `FindRoot`.

9. À quelle distance du pôle nord et à quelle altitude se trouve alors la particule ?

10. Représentez dans le plan (r, z) les iso-contours de $T(r, z)$ à l'aide de la commande `ContourPlot`. Pour obtenir un résultat en couleurs parlant d'un point de vue thermique, on vous suggère d'utiliser l'option `ColorFunction->"Temperature"` de `ContourPlot`. Superposez à ce graphe la projection de la trajectoire de la particule dans le plan (r, z).

    Indication : pour cette superposition utilisez la commande `Show` :

```
17  gisoT= ContourPlot[T[r,z],...]
18  gtrajrz= ParametricPlot[{r[t],z[t]},...]
19  gf= Show[gisoT,gtrajrz]
```

11. *Question subsidiaire aux physiciens* : critiquez d'un point de vue physique la validité de ce modèle et surtout du calcul de trajectoire précédent.

Commentaires :

L'existence d'un vortex polaire du type précédent est intimement liée à celle du ***trou d'ozone***, à cause de couplages thermo-hydro-physico-chimiques. En particulier certaines réactions sont très sensibles à la température, d'où l'intérêt d'études du type précédent, même si dans la réalité les choses sont plus compliquées...

TP 3

Traitement d'un signal sonore

3.1 Introduction

Dans ce dernier TP Mathematica, nous allons illustrer les capacités du logiciel et voir la manière de s'en servir en situation quasi-réelle.

Pour cela, nous allons traiter un échantillon de signal sonore contenu dans un fichier au format `wav`. Ce travail relève de l'analyse numérique et du traitement du signal.

3.2 Récupération du fichier de données

Pour récupérer le fichier de donnée, télécharger le fichier bonjour.wav sous Arche dans la section TP Mathematica E&F, définir votre répertoire de travail à l'aide de la commande SetDirectory[ ] et sauvegarder-le sous votre répertoire de travail.

1. Sous Mathematica, on va récupérer les données correspondant à l'échantillon sonore contenu dans le fichier `TP_math.wav`.

```
data=Import["TP_math.wav","Data"];
sr=Import["TP_math.wav","SampleRate"];
ListPlay[data[[1;;2]],SampleRate->sr]
```

2. Expliquer la structure des données obtenue sous Mathematica, c'est-à-dire ce que contient data et sr ainsi que leur type (liste, vecteur, nombre entier, etc ...). Pour accéder à la taille d'un tableau, on peut utiliser la fonction :

```
Dimensions[data]
```

3.3 Traitement du signal

3.3.1 Transformée de Fourier

1. Afin de traiter le signal, nous allons séparer les deux canaux sonores (c'est en stéréo !). N'oublier pas les « ; » car les tableaux sont grands.

```
ch1=data[[1]]; ch2=data[[2]];
```

2. Transformons le son stéréo en son monophonique. Pour cela, faites une moyenne arithmétique des deux canaux sonores. Écouter le résultat sous Mathematica et mettre les données dans une variable appellée `sonmono`.

   Pour sauvegarder sous forme de fichier wav, taper la commande :

```
6  Export["sonmono.wav",Sound[SampledSoundList[sonmono, sr ]]];
```

3. Vous pouvez vérifier que le fichier `sonmono.wav` a bien été créé et qu'il est lisible par n'importe quel logiciel de lecture audio (double clic sur le fichier).
4. Maintenant, nous allons nous intéresser au signal contenu dans `sonmono`. Nous pourrions faire de même pour `ch1` et `ch2` pour conserver un son stéréo.
   En vous aidant de l'aide en ligne, faites une transformée de Fourier du signal (voir `Fourier`) et mettez-la dans un tableau nommé `sonF`.
5. Générer un tableau nommé `f` contenant les fréquences correspondant à la transformée de Fourier.
6. Tracer le spectre de puissance en mettant la fréquence en abscisse. Pour cela, utiliser `ListLinePlot`.
7. Regénérer le signal sonore avec la fonction `InverseFourier` et mettez-le dans une variable nommée `sonIF`. On pourra vérifier la correspondance du signal d'origine et du signal reconstitué en affichant le maximum de la différence entre eux. Lire le fichier sonIF, qu'observe t'on ? En quoi l'utilisation de la fonction `Re` permettrait la lecture de `sonIF` ?

3.3.2 Compression des données

1. Afin de réduire la quantité de donnée à stocker, on va procéder à un traitement très simple du spectre : on ne va conserver que les fréquences dont l'énergie représente plus de $1/100^{\text{ème}}$ du pic maximal du spectre de puissance (les autres seront mises à zéro). On ne prendra pas en compte la valeur moyenne pour la recherche du pic d'intensité maximale.
2. Reconstituer le signal sonore à partir du spectre simplifié obtenu après traitement.
3. Pour sauvegarder le spectre simplifié, on n'enregistre que les composantes de Fourier non nulles. Afin de reconstituer un spectre, on doit aussi sauvegarder la fréquence (l'indice de la composante). Les composantes étant complexes, elles prennent la place de deux nombres réels en mémoire. On considèrera que l'indice (la fréquence) prend la place d'un nombre réel. Ainsi, chaque pic du spectre sauvegardé prend la place de trois nombres réels en mémoire. Enfin, comme le signal est une suite de nombres réels, on sait que la deuxième moitié du spectre est le conjugué complexe de la première moitié (en dehors de la valeur moyenne, bien entendu). Il suffit donc de ne sauvegarder que la moitier du spectre obtenu.
4. En comptant le nombre de composantes non nulles du spectre obtenue après traitement, donner le gain en mémoire par rapport au stochage du tableau de réels `sonmono`. Donner le gain en pourcentage.
5. Le son est assez déformé. En effet, le traitement est très (trop) basique. Cependant, régler le seuil à $1/10000^{\text{ème}}$ au lieu de $1/100^{\text{ème}}$ améliore notablement la qualité du son. Quelle est alors le gain ?
6. Pour générer un tableau ne contenant que les composantes non nulles après traitement, utiliser la commande `AppendTo` après avoir créé un tableau vide. Mettre chaque composante non nulle du spectre dans un tableau nommé `soncomp` ainsi que son indice dans le spectre initial. Exemple d'utilisation :

```
7  toto={};
8  n=10;
9  For[i=1,i<n+1,i++,
10 ____AppendTo[toto,{i, i^2}];
11 ];
```

7. Reconstituer un spectre complet en ajoutant des composante nulles à partir de `soncomp`. Par transformation inverse de Fourier, reconstituer le signal sonore.

3.3.3 Qualité du signal après compression

En écoutant le son, on peut se faire une idée de la qualité (médiocre!) du son obtenu après compression. Cependant, on peut utiliser Mathematica pour donner des résultats permettant de quantifier les pertes.

1. Sachant que l'énergie E contenu dans un signal $x(t)$ peut être calculer grâce à sa décomposition en série de Fourier $X(f_i)$ par :

$$E = \sum_{i=1}^{n} |X(f_i)|^2 \tag{3.1}$$

   Donner en pourcentage la quantité d'énergie perdue après traitement (prendre le seuil de $1/100^{\text{ème}}$).

2. On pourra afficher un graphique représentant le signal d'origine et la différence entre ce dernier et le signal reconstitué. En observant ce graphique et à l'aide de la quantité d'énergie perdue, que pouvez-vous dire sur la qualité du son reconstitué avec un seuil à $1/100^{\text{ème}}$ et à $1/10000^{\text{ème}}$?

3.3.4 Tentative d'amélioration

Afin d'essayer d'améliorer le rendu sonore, on va utiliser le fait que l'oreille humaine perçoit les sons dont la fréquence varie de 20 Hz à 20 kHz et que le spectre de la voix humaine couvre les fréquences jusqu'à 4000 Hz.

1. Générer un tableau contenant les composantes de Fourier allant de 20 Hz à 4000 Hz. Ne pas oublier de mettre l'indice de la composante dans le tableau. Quel est le gain ?
2. Reconstituer le signal sonore à partir du spectre compris entre 20 Hz et 4000 Hz.
3. On voit que dans notre cas, le gain n'est pas intéressant. On va donc encore filtrer le spectre de puissance en supprimant les pics inférieurs à $1/10000^{\text{ème}}$ du maximum des pics entre 20 Hz et 4000 Hz. Quel est alors le gain par rapport au tableau d'origine `sonmono` ? Quelle est la perte d'énergie par rapport au signal d'origine ?