Cours-TD n° 3 : Éléments de langage Mathematica II

Table des matières

1 Préparations pour ce TD

1.1 Terminer TD 2

Si vous n'avez pas eu le temps de terminer le TD 2, alors profitez lors de cette séance - moins chargée - pour terminer le TD 2.

1.2 Raccourcis clavier

N'oubliez pas le feuille avec les **raccourcis clavier** et essayez d'utiliser la souris le moins possible.

1.3 Exemples de code Mathematica

Pour bien apprendre les notions traitées ici, il est très imortant de tester **et comprendre** les exemples de code Mathematica donnés ici. Il ne suffit pas juste de taper ces lignes de code sans chercher à les comprendre.

Tous les exemples de code Mathematica montrés dans l'énoncé ne contiennent que des cellules de type Input. C'est pour cela que c'est indispensable de tester ces exemples soi-même avec Mathematica. **Chaque ligne d'un exemple est une nouvelle cellule** de type Input et doit être évaluée dans l'ordre. Les exemples peuvent être copié&collé (Ctrl+C puis Ctrl+V) dans Mathematica depuis la version PDF de l'énoncé présent, mais attention de séparer la commande Quit dans une cellule à part. En général si les commandes ne sont pas trop longues, il est conseillé de ne pas utiliser le copier&coller, mais de taper soi-même les commandes. Cela aide à l'apprentissage de ces commandes et de plus le copier&coller ne fonctionne pas dans certains cas.

1.4 Exercices

Les exercices sont indiqués avec un numéro sur le bord gauche de l'énoncé. Pensez à indiquer ce numéro dans votre notebook d'une manière bien visible, par exemple en utilisant le type de cellule "Section" (Alt+4).

1.5 Pour aller plus loin

Ceux qui auront déjà terminé tous les exemples et exercices avant la fin de la séance peuvent étudier quelques "Problem" au choix du livre de Hazrat (voir sur SAKAI sous le dossier "livres") et d'essayer de résoudre quelques "Exercice" (solutions à la fin du livre).

2 Transformation d'expressions

2.1 Expand, Factor

2.1.1 Expand

Il y a plusieurs façon d'écrire une expression algébrique, par exemple $(x+1)^2$ peut être écrit comme x^2+2x+1. Cette opération est fait par `Expand` :

```
Expand[(x+1)^2]
```

On voit que Mathematica trie les termes en puissance croissante.

`Expand` ne change pas les sous-expressions, mais `ExpandAll` le fait :

```
Expand[Sqrt[(x+1)^2]]
ExpandAll[Sqrt[(x+1)^2]]
```

Si on veut développer qu'une partie de l'expression, on peut spécifier quelle partie doit être développée. Tous les termes ne contenant pas cette partie restent inchangés :

```
Expand[(a + b)^2 * (1 + x)^2, x]
Expand[(1 + x)^2 + (2 + x)^2, 1 + x]
```

2.1.2 Factor

La fonction inverse d'`Expand` est `Factor` :

```
Factor[x^2+2*x+1]
```

Certaines expressions ne sont pas factorisées par `Factor` :

```
Factor[2 + 2 Sqrt[2] x + x^2]
```

Il faut ajouter une option :

```
Factor[2 + 2*Sqrt[2]*x + x^2, Extension -> Automatic]
```

La recherche automatique de coefficients ne marche pas dans tous les cas, il faut alors spécifier manuellement le ou les coefficients.

```
Factor[1 + x^4]
Factor[1 + x^4, Extension -> Automatic]
Factor[1 + x^4, Extension -> Sqrt[2]]
```

1. Exercice : Factoriser $x^2 - 3$ avec Mathematica.

2.1.3 ExpandDenominator, ExpandNumerator

Regarder dans l'aide de Mathematica ce que font les fonctions `ExpandDenominator` et `ExpandNumerator`.

2. Soit :

$$f(x) = \frac{(x+3)(x-1)^2}{(x^2+1)(x+5)^2}$$

Appliquer les fonctions `Expand`, `ExpandAll`, `ExpandDenominator` et `ExpandNumerator` à la fonction $f(x)$ et observer les différences dans les résultats.

2.1.4 Together, Apart, Cancel

Regarder dans l'aide de Mathematica ce que font les fonctions `Together`, `Apart` et `Cancel`.

3. Chercher les fonctions Mathematica qui transforment (en une ou plusieurs étapes) $f(x)$ en $g(x)$ avec

$$f(x) = \frac{1}{x^2-16} - \frac{x+4}{x^2-3x-4}$$

$$g(x) = \frac{-15-7x-x^2}{-16-16x+x^2+x^3}$$

2.2 Simplify, FullSimplify

Pour simplifier une expression algébrique en Mathematica, il existe deux fonctions : `Simplify` et `FullSimplify`. `Simplify` applique uniquement les règles algébriques standard de transformation, tandis que `FullSimplify` applique toutes les règles intégrées dans Mathematica. Les deux fonctions essaient de trouver l'expression la plus courte possible.

2.2.1 Simplify

`Simplify` essaie de trouver l'expression la plus courte possible. Si par exemple la factorisation donne une expression plus courte, alors `Simplify` fait la même chose que `Factor` :

```
Factor[x^2 + 2*x + 1]
Simplify[x^2 + 2*x + 1]
```

Tandis que si la factorisation donnerait une expression plus longue, l'expression n'est pas changé par `Simplify` :

```
Factor[x^10 - 1]
Simplify[x^10 - 1]
```

Souvent les autres fonctions de Mathematica ne fournissent pas automatiquement l'expression la plus simple. `Simplify` est alors souvent utilisée pour simplifier les résultats des autres fonctions, comme montré ici avec les fonctions `Integrate` et `D` (voir l'aide de Mathematica (touche F1) pour ces deux fonctions, ainsi que pour `Clear`) :

```
Clear[x,f]
f[x_] := 1/(x^4 - 1)
f[x]
integrale = Integrate[f[x], x]
derivee = D[integrale, x]
Simplify[derivee]
```

2.2.2 FullSimplify

Comme la fonction `FullSimplify` teste une plus grande gamme de transformations que `Simplify`, elle permet de simplifier des expressions que `Simplify` n'arrive pas à simplifier :

```
Clear[x,f]
f[x_] := (Sqrt[(2 + x)/(7 + x)]*
          Sqrt[(3 - x^2)/(25 - x^2)]*
          Sqrt[(5 + x) (2 - x)])/
          (Sqrt[4 - x^2]*Sqrt[3 - x^2])
f[x]
Simplify[f[x]]
FullSimplify[f[x]]
```

`FullSimplify` peut prendre beaucoup de temps de calcul surtout pour des expressions très grandes, d'où l'intérêt de la fonction `Simplify` qui est plus rapide. Si même `Simplify` prend trop de temps de calcul, alors il faut essayer de guider manuellement la transformation recherchée.

Un autre exemple sur les fonctions trigonométriques :

```
Clear[x,f]
Sin[x]^2 + Cos[x]^2
Simplify[Sin[x]^2 + Cos[x]^2]
f[x_] := Tan[x]^2*(3 + 3*Tan[x]^2 + Tan[x]^4)
f[x]
Simplify[f[x]]
FullSimplify[f[x]]
```

2.2.3 Simplify avec ShowSteps

Sur certains résultats de `Simplify` on peut avoir les étapes intermédiaires avec Wolfram Alpha.

4. Reprendre la fonction `ShowSteps` (voir énoncé TD 2) pour obtenir les étapes intermédiaires de la simplification de

$$\frac{k}{k+1} + \frac{1}{(k+1)(k+2)}$$

2.3 Transformer des expressions trigonométriques

2.3.1 Introduction

Les fonctions de transformation d'expression algébrique comme `Expand` ou `Factor` laissent les expressions trigonométriques inchangées, par exemple :

```
Expand[Sin[x + y]]
```

Pour inclure les transformations d'expressions trigonométriques on utilise les fonctions qui ont un "`Trig`" en plus dans leur nom, comme par exemple `TrigExpand` ou `TrigFactor`. Ces fonctions transforment à la fois des expressions algébriques et trigonométriques :

```
TrigExpand[Sin[x + y]]
TrigExpand[(x+1)^2]
TrigFactor[Sin[x]^2 + Tan[x]^2]
TrigFactor[x^2 + 2*x + 1]
```

Les trois fonctions `Simplify, FullSimplify` et `ComplexExpand` (voir en bas) transforment aussi à la fois des expressions algébriques et trigonométriques, donc il n'existe pas une version "`Trig`" de ces trois fonctions.

Les formules d'identités trigonométriques peuvent être trouvées sur Wikipedia en français :

http://fr.wikipedia.org/wiki/Identit%C3%A9_trigonom%C3%A9trique

Mais il y a plus de détails sur la page équivalente en anglais ou encore en allemand ;-) (cliquer sur **English** à gauche sous **Autres langues**) :

http://en.wikipedia.org/wiki/List_of_trigonometric_identities

http://de.wikipedia.org/wiki/Formelsammlung_Trigonometrie

2.3.2 TrigExpand

`TrigExpand` développe une expressions en deux étapes :

1. Séparation des sommes et multiples entiers des arguments d'une fonction trigonométrique, par exemple :

```
TrigExpand[Sin[2*x]]
TrigExpand[Sin[x+y]]
```

2. Ensuite développement du produit de fonctions trigonométriques en sommes de puissances :

```
Quit
a = TrigExpand[Sin[2*x]]
b = TrigExpand[Cos[2*y]]
c = a*b
Expand[c]
TrigExpand[Sin[2*x]*Cos[2*y]]
```

Dans l'exemple au dessus les deux commandes `Expand[c]` et `TrigExpand[Sin[2*x]*Cos[2*y]]` donnent le même résultat, car aucune identité trigonométrique y est appliquée à la deuxième étape de `TrigExpand`. Un exemple où une identité trigonométrique est appliquée à la deuxième étape :

```
Expand[Sin[x]^2*Cos[y]]
TrigExpand[Sin[x]^2*Cos[y]]
```

2.3.3 TrigFactor

La fonction `TrigFactor` essaie de factoriser une expression trigonométrique le plus possible en appliquant les identités trigonométriques. Elle peut changer les arguments des fonctions trigonométriques. Exemple :

```
TrigFactor[Cos[y]*Sin[x] + Cos[x]*Sin[y]]
TrigFactor[Sin[x] + Cos[x]]
```

2.3.4 TrigReduce

La fonction `TrigReduce` transforme des expressions trigonométriques en combinant les arguments ou en appliquant les règles d'identité d'angles multiples. Exemples :

```
TrigReduce[2*Sin[x]*Cos[y]]
TrigReduce[2*Cos[x]^2]
```

En général `TrigReduce` est la fonction inverse de `TrigExpand` :

```
Clear[x,y]
a=Sin[2*x]
b=TrigExpand[a]
c=TrigReduce[b]

a=Sin[x+y]
b=TrigExpand[a]
c=TrigReduce[b]
```

5. La fonction `TrigFactor` peut fournir dans certains cas le résultat inverse de `TrigExpand`. Tester cela sur les deux exemples d'en haut : `Sin[2*x]` et `Sin[x+y]`. Dans lesquels des deux cas `TrigFactor` donne le résultat inverse de `TrigExpand` et pourquoi ?

6. Appliquer `TrigReduce` sur $f(x, y, z) = \sin^2(x)\cos(y)\sin(z)$ et `TrigFactor` sur le résultat. Commenter.

7. Appliquer `TrigExpand` sur $f(x, y, z) = \sin^2(x)\cos(y)\sin(z)$ et `TrigReduce` sur le résultat. Commenter. Trouver la fonction inverse de `TrigExpand` dans ce cas présent.

8. Comparer et commenter les trois résultats des fonctions `TrigExpand`, `TrigReduce` et `TrigFactor` appliquées à $f(x) = \sin^2(x) + \tan^2(x)$.

2.3.5 Simplify

Pour retrouver l'expression de départ après un `TrigExpand` et `TrigReduce` ou `TrigFactor` successif, il peut être nécessaire d'utiliser `Simplify` :

```
Clear[x,f,a,b,c]
f[x_] := Sin[2*x]*Cos[2*y]
f[x]
a = TrigExpand[f[x]]
b = TrigFactor[a]
c = TrigReduce[a]
Simplify[b]
Simplify[c]
```

On peut facilement vérifier des identités trigonométriques avec `Simplify` et `==`, comme par exemple :

```
Simplify[Cos[a + b] == Cos[a]*Cos[b] - Sin[a]*Sin[b]]
```

9. Vérifier les identités trigonométriques suivantes :

$$\sin(a + b) = \sin(a)\cos(b) + \cos(a)\sin(b)$$

$$(1 - \cot(a))^2 + (1 - \tan(a))^2 = (\sec(a) - \csc(a))^2$$

2.4 Assumptions

2.4.1 Simplify avec assumptions

Mathematica ne simplifie pas automatiquement $\sqrt{x^2}$ en x car ceci n'est possible que pour $x \geq 0$:

```
Simplify[Sqrt[x^2]]
```

Mais on peut donner à `Simplify` la supposition (assumption en anglais) que $x \geq 0$:

```
Simplify[Sqrt[x^2], x >= 0]
```

Dans cet exemple on aurait pu aussi utiliser la fonction `PowerExpand` à la place de `Simplify` :

```
PowerExpand[Sqrt[x^2]]
```

On peut aussi donner plusieurs suppositions avec les connecteur logiques (`&&` veut dire ET, `||` veut dire OU) :

```
expression=2*a+2*Sqrt[a-Sqrt[-b]]*Sqrt[a+Sqrt[-b]]
Simplify[expression]
Simplify[expression, a > 0 && b > 0]
```

On peut aussi définir une borne inférieure et supérieure à la fois sans utiliser `&&` :

```
Simplify[ArcSin[Sin[x]], -Pi/2 < x < Pi/2]
```

2.4.2 Element

Comme supposition on peut aussi définir le domaine d'une variable avec la fonction `Element`. Par exemple pour le cas montré en haut on peut aussi juste définir que x est une valeur réelle et non complexe, ce qui donne :

```
Simplify[Sqrt[x^2], Element[x, Reals]]
```

Définir une variable n entier peut être utile pour les fonctions trigonométriques :

```
Simplify[Sin[x + 2*n*Pi], Element[n, Integers]]
Simplify[Cos[n*Pi/2-x], Element[(n-1)/4, Integers]]
```

On peut aussi combiner `Element` avec d'autres conditions :

```
Simplify[Log[x^r]]
Simplify[Log[x^r], x > 0]
Simplify[Log[x^r], Element[r, Reals]]
Simplify[Log[x^r], x > 0 && Element[r, Reals]]
```

10. Regarder dans l'aide de Mathematica sous **guide/AssumptionsAndDomains** ou encore dans **tutorial/UsingAssumptions** pour trouver la liste des **Domains**. Reporter cette liste dans votre compte rendu et expliquer à quoi correspond chaque domaine. En regardant le sous-point **Properties&Relations** de l'aide sur **Algebraics**, définir quels domaines **Algebraics** contient et ne contient pas, ainsi que dans quels domaines **Algebraics** est contenu ou pas. Plus d'infos ici : http://fr.wikipedia.org/wiki/Nombre_alg%C3%A9brique

3 Nombres complexes

3.1 Introduction

Pour écrire un nombre complexe comme $z = a + ib$ dans Mathematica, on écrit un grand I :

```
z = a + I*b
```

Pour afficher le grand I comme dans le Output on appuie sur la touche **Echap** puis deux fois un petit i et pour terminer encore **Echap**, en résumé :

Echap ii **Echap**

Certains calculs peuvent donner des valeurs complexes :

```
Sqrt[-4]
```

On peut faire aussi des calculs avec nombres complexes :

```
(4+3*I)/(2-I)
```

3.2 Re, Im, Abs et ComplexExpand

Par défaut Mathematica considère toutes les variables dans le domaine des complexes. C'est pour cela qu'on obtient des résultats un peu bizarre avec les trois fonctions `Re, Im, Abs`, car Mathematica considère `a` et `b` en étant elles mêmes des variables complexes :

```
z = a + I*b
Re[z]
Im[z]
Abs[z]
```

Une solution pour retrouver les résultats habituelles est d'utiliser la fonction `ComplexExpand` qui considère toutes les variables d'une expression en étant réelles, sauf celles qu'on lui spécifie explicitement :

```
z = a + I*b
ComplexExpand[Re[z]]
ComplexExpand[Im[z]]
ComplexExpand[Abs[z]]
```

Deux exercices issus de l'électrocinétique tel qu'elle était encore enseignée à l'UPMC en L1 jusqu'en 2013 (UE LP103). Il s'agit de transformer des expressions d'impédance ou admittance complexe d'un circuit RLC en régime sinusoïdal :

11. En utilisant les fonctions `Re, Im` et `ComplexExpand` développer le nombre complexe z en $Re(z) + i * Im(z)$:

$$z = iC\omega + \frac{1}{iL\omega + R}$$

12. En utilisant les fonctions `Simplify`, `FullSimplify`, `Abs` et `ComplexExpand` montrer que

$$|z| = \sqrt{\frac{C^2w^2\left(L^2w^2 + R^2\right) - 2CLw^2 + 1}{L^2w^2 + R^2}} = \sqrt{\frac{Cw^2\left(L\left(CLw^2 - 2\right) + CR^2\right) + 1}{L^2w^2 + R^2}}$$

4 Sommes, Produits et Développement limité

4.1 Sum et Product

La fonction `Sum` a la même syntaxe que la fonction `Table` (voir TD 2), voici quelques exemples :

```
Table[x^i/i, {i, 1, 7}]
Sum[x^i/i, {i, 1, 7}]

Table[x^i/i, {i, 7}]
Sum[x^i/i, {i, 7}]

Table[x^i/i, {i, 1, 5, 2}]
Sum[x^i/i, {i, 1, 5, 2}]
```

La fonction `Product` a également la même syntaxe que `Sum` ou `Table` :

```
Product[x + i, {i, 1, 4}]
```

Contrairement à `Table` on peut définir des bornes infinies et obtenir une valeur exacte :

```
Sum[1/i^4, {i, 1, Infinity}]
```

Ou encore une valeur approchée si une valeur exacte ne peut pas être calculée :

```
somme = Sum[1/(i! + (2 i)!), {i, 1, Infinity}]
N[somme]
```

13. Écrire la fonction $ep(n) = 1 + \frac{1}{1} + \frac{1}{2!} + \ldots + \frac{1}{n!}$ et calculer la valeur approchée de $ep(10)$ et la valeur exacte de $ep(\infty)$. A quoi correspond la fonction ep ?

14. Tracer $ep(n)$ pour $1 \leq n \leq 10$, n entier, avec les fonctions `Table` et `ListLinePlot`.

4.2 Développement limité

La fonction `Series`$[expr,\ \{x, x_0, n\}]$ fournit le développement limité (= série de Taylor) d'une fonction autour du point x_0 jusqu'à l'ordre n. Exemples :

```
Series[Exp[x], {x, 0, 4}]
Series[f[x], {x, 0, 3}]
```

Il y aura une séance spécialement dédiée au développement limité.

5 Introduction aux vecteurs et matrices

5.1 Vecteurs

Les vecteurs sont représentés comme des listes (voir TD 2) en Mathematica :

```
v = {x, y, z}
u = 2*v
u*v
u.v
Norm[v]
Normalize[v]
```

Le produit scalaire s'écrit soit avec un point, soit avec la fonction `Dot`. Le produit vectoriel s'écrit soit avec la fonction `Cross` soit avec une croix qu'on obtient avec :
Echap cross **Echap**

```
v = {x, y, z}
u = 2*v
Dot[u,v]
u.v
Cross[u,v]
u x v
```

Voir l'aide de Mathematica sous **guide/OperationsOnVectors** pour plus d'opérations sur les vecteurs.

5.2 Matrices

Les matrices sont représentées comme une liste de listes en Mathematica. Exemple :

```
mat = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
mat // MatrixForm
```

On peut générer une matrice avec la fonction `Table` (voir TD 2) :

```
mat = Table[x^i + x^j, {i, 1, 4}, {j, 1, 3}]; mat // MatrixForm
```

Voir l'aide de Mathematica sous **tutorial/VectorsAndMatrices** pour plus d'information sur les matrices et vecteurs.

6 Pour aller plus loin

Ceux qui auront déjà terminé le TD 2 et 3 avant la fin de la séance peuvent étudier quelques "Problem" au choix du livre de Hazrat (voir sur SAKAI sous le dossier "livres") et d'essayer de résoudre quelques "Exercice" (solutions à la fin du livre).