Chapitre 1
LA MAITRISE DE MATHEMATICA

Présenter succintement Mathematica est un défi irréaliste : le manuel de référence de l'auteur, Stephen Wolfram, relatif à la version 3, (actuellement, la version commercialisée porte le numéro 5), représentait un pavé de 1500 pages. A ce jour, aucune édition papier n'est fournie avec ce brillant logiciel, la documentation est intégrée.

Mathematica, logiciel complet, ne se réduit pas à un puissant système de calcul formel, il rassemble dans un environnement unifié les fonctions de solveur algébrique et numérique, de grapheur, d'éditeur de documents scientifiques, complété d'un langage de programmation multi-paradigmes.

En un peu plus d'une centaine de pages, illustrées de nombreux exemples et d'une liste d'exercices corrigés, nous dégageons dans ce chapitre, les connaissances nécessaires pour utiliser rationnellement Mathematica. Nous nous concentrerons sur les caractéristiques générales du système. Au niveau de la programmation, nous vous inciterons systématiquement à abandonner les méthodes héritées d'autres langages.

INTRODUCTION

Historique de Mathematica

Mathematica, système de calcul formel, apparu en 1988 connaîtra de profondes évolutions que nous résumons succintement.

• Dès la version 1.0, apparaissent les concepts clés, fonctions, expressions, listes, module graphique, notion de Notebook, langage extensible avec les packages (fichiers de commandes que l'on importe en cas de besoin dans la feuille de calcul), programmation suivant divers paradigmes.
Ces concepts majeurs imposent Mathematica comme un système de calcul formel novateur et puissant. Quelques lignes de programme suffisent pour résoudre des problèmes d'une réelle complexité.
Les évolutions de Mathematica ne remettront jamais en cause l'investissement intellectuel réalisé. Les concepts fondamentaux, piliers de la programmation seront conservés. Le nombre de fonctions Mathematica (primitives) gonflera pour atteindre des milliers, l'optimisation permanente de ces fonctions rendra l'exécution des tâches plus rapide et fluide. A chaque nouvelle version de Mathematica, les programmes anciens se convertissent de façon transparente à l'utilisateur en programmes compatibles avec la nouvelle version.

• La version 2.0 développe la notion de noyau (Kernel) et d'interface (partie frontale ou Front End en anglais). La compilation des fonctions voit le jour.
L'idée de séparer la partie interface (présentation, fenêtres…) du noyau (qui contient les librairies chargées d'effectuer les calculs) ne coulait pas de source. A l'époque, les exigences de Mathematica en puissance de processeur et de mémoire se révélaient coûteuses.
Cette séparation permet de n'utiliser que l'interface si l'objectif est d'écrire un rapport, un mémoire, un chapitre de livre, en minimisant l'espace mémoire.
Le noyau permet de réaliser les calculs, de tester les programmes. Le caractère universel de Mathematica, permet l'écriture des programmes sur un poste bureautique, puis l'exécution du code sur des machines plus puissantes, stations de travail…

• La version 3.0 présente une évolution majeure, en termes d'optimisation, et d'interface. Les palettes flottantes avec tous les symboles mathématiques usuels font leur apparition. L'utilisateur dispose ainsi d'outils évolués et puissants pour éditer des travaux scientifiques, rapports, livres…

• Les versions 4, puis la version 5.0 apportent leur lot de nouveautés (se référer au menu Help pour le détail) : essentiellement la simplification d'expressions avec des conditions.

L'optimisation des fonctions se révèle souvent très spectaculaire.
En résumé, avec la version 5 très aboutie, le monde scientifique dispose d'un excellent outil de calcul, de recherche et d'édition.

Les apports du calcul formel

Mathematica, ainsi que Maple sont des logiciels de calcul symbolique très performants, permettant d'apprendre et de faire des Mathématiques autrement, jusqu'au très haut niveau.
Le concept de calcul formel permet :

- de s'affranchir des calculs fastidieux,
- d'expérimenter,
- de se concentrer sur l'analyse des thèmes proposés,
- de réfléchir à la portée des outils dont on dispose,
- de critiquer les résultats obtenus.

La contre-partie réside dans une relative complexité d'utilisation, une formation rigoureuse s'impose : sinon pointe le risque d'être un éternel débutant.
Les concepts fondamentaux du langage sont puissants mais parfois difficiles, les primitives du système nombreuses, les styles de programmation variés et peut-être déroutants : le débutant livré à lui-même progressera peu, et au pire se découragera. Nous avons donc choisi de démarrer par l'exposé condensé mais rigoureux, de toutes les notions fondamentales à la bonne compréhension de Mathematica, suivi des applications en arithmétique.
Nous exposerons ultérieurement les applications en algèbre, algèbre linéaire et multilinéaire, géométrie et analyse, domaines dans lesquels le calcul formel apporte un éclairage nouveau et très intéressant.

Le mode interactif est de mise en calcul formel : l'utilisateur écrit ses lignes de commandes dans des cellules de type "Input", les exécute, observe, analyse ou critique les sorties. Les résultats obtenus sont récupérables pour être exploités par des commandes futures, etc…

Mathematica langage de programmation

Mathematica ne se limite pas à une bibliothèque impressionnante de fonctions, mais il permet de programmer sous différents styles (paradigmes) indépendamment de l'ordinateur.
Nous étudierons et comparerons ces paradigmes à longueur d'ouvrage : la programmation sous Mathematica est passionnante, tant les concepts du langage sont puissants et cohérents.
Le nombre élevé de primitives en calcul symbolique empêche l'utilisateur de les connaître toutes, leur nombre grandira à chaque évolution du langage. Une exhaustive aide en ligne permet de se documenter sur telle primitive, en sachant dans quelle catégorie la retrouver.

I. LES NOTIONS DE BASE

Dans ce paragraphe, nous allons illustrer les notions de base que le lecteur devra impérativement dominer, afin de tirer le meilleur profit de Mathematica.

1. Les diverses notions d'égalité en calcul formel

Sous Mathematica, la notion d'égalité est multiforme. Il existe quatre types d'égalités, deux types réservés à l'affectation, et deux autres à la comparaison de variables.

1.1 L'affectation

• En calcul formel, une variable est dite libre si elle n'a reçu aucune affectation.
Pour libérer une variable nous écrirons :

```
Clear[maVariable]      ou encore :
maVariable=.
```

• Pour réaliser une affectation, nous disposons de deux moyens :

(1) L'affectation immédiate, `maVariable=valeur` (par exemple $x = 5$).

(2) L'affectation retardée, `maVariable := valeur` (exemple $x := 5$).

Dans le premier cas, la variable x reçoit la valeur 5 dès la validation de la ligne de commande. Lors d'une affectation différée du type `x := 5`, l'affectation se réalisera seulement à l'appel de la variable x, et non après l'écriture de la commande.

• Dans les définitions de fonctions, l'affectation différée est beaucoup plus utile que l'affectation immédiate. Si vous donnez à une expression sa valeur définitive, utilisez l'affectation immédiate. En cas de doute, préférez l'affectation différée.

1.2 La comparaison

Comment tester en calcul formel l'égalité de deux variables a et b ?
L'égalité simple étant réservée à l'affectation immédiate, ce type se comparaison se réalise en écrivant une double ou triple égalité :

(1) La comparaison `a == b` renvoie (si possible) un booléen (`True`, `False`).

(2) La comparaison formelle `a === b` *force* l'évaluation du booléen.

Etudions la différence entre les deux types de comparaison, avec l'exemple suivant :

```
Clear[a]
{a == 3, a === 0}
```

```
{a == 3, False}
```

Mathematica ne sait pas évaluer le booléen `a == 3`, puisque la variable *a* n'a reçu aucune valeur, il renvoie la question, par contre le booléen `a === 0` est faux. Lorsque l'évaluation est impossible, la triple égalité renvoie la valeur `False`.
Il y a une grande importance en pratique à comprendre un principe fondamental : une variable formelle libre n'est égale à aucune valeur. Ainsi, Mathematica traîte les expressions mathématiques constituées avec des variables formelles libres, sans se soucier de leur existence.
Dans une expression telle que $\frac{a^2+b^2}{a^2-b^2}$, le dénominateur est supposé non nul. Par contre, l'évaluation de cette expression provoquera un avertissement si $a = b$ ou $a = -b$:

```
expr := (a^2 + b^2)/(a^2 - b^2)

a := 5; b := -5;
expr
```

```
Power::infy : Infinite expression 1/0 encountered. Plus…

ComplexInfinity
```

---

2. Les règles de substitution

2.1 Généralités sur les règles de substitution

• Mathématica produit souvent des sorties sous forme de règles afin d'éviter l'affectation. Une liste de règles s'écrit sous la forme : $\{x \to \cdots,\ y \to \cdots, \cdots\}$.
L'un des problèmes du débutant consistera à réutiliser ces règles, bien souvent en les convertissant en résultats explicites, nous illustrons ce point plus loin.
• L'opérateur de substitution le plus fréquent est : `/.`
Soit *expr* une expression de trois variables *x*, *y* et *z* :

```
expr /. {x → 3, y → 5, z → 1}
```

représente la valeur de *expr* en remplaçant *x* par 3, *y* par 5 et *z* par 1.
La flèche s'obtient dans la palette BasicInput, ou bien on tape le signe `-` suivi de `>`.
Note très importante : la valeur formelle de *expr* n'est pas modifiée et les variables *x*, *y*, *z* ne reçoivent pas d'affectation.

```
expr = x^2 - y^2 + 2 z^4;
expr /. {x → 3, y → 5, z → 1}
```

```
-14
```

• Vous le constaterez par la pratique de Mathematica, l'affectation systématique de valeurs aux variables présente des inconvénients : lors d'une session complète, on ne sait plus très bien quelle variable est affectée, ce qui risque de poser des diffultés imprévues. Illustrons notre propos d'un exemple simple :

```
x := 2
p := x^2 - 5 x + 4
p /. x → 1
```

```
-2
```

La règle de substitution ne produit pas le résultat escompté (0), car lors de l'évaluation de *p*, Mathematica évalue *x* (affectation différée) et remplace donc *x* par 2. Evidemment, dans ce cas on comprend très vite la situation, mais imaginez que *x* ait été affecté bien avant la définition de *p*…Voici la bonne démarche :

```
Clear[x]
p /. x → 1
```

```
0
```

2.2 Les primitives ReplaceAll et ReplaceRepeated

• Plus généralement, soit *expr* une expression quelconque, et *regle* une liste de règles de transformation :

`expr /. regle` applique les règles à chaque partie de *expr*.
`expr //. regle` applique les règles jusqu'à ce que le résultat soit invariant.

`/.` est la contraction (ou notation infixée) de la primitive `ReplaceAll`, tandis que `//.` est celle de `ReplaceRepeated`.
Exemple :

```
expr := x^2 + y^2
expr1 = expr /. {x → a - b, y → 2 x}
expr2 = expr //. {x → a - b, y → 2 x}
```

```
(a - b)^2 + 4 x^2
```

```
5 (a - b)^2
```

Dans le dernier résultat, Mathematica remplace x par $a - b$, puis y par $2\,x$ et enfin de nouveau x par $a - b$, pour produire le résultat final $5\,(a - b)^2$.

Les règles de substitution permettent de laisser les variables libres (non affectées), ce qui est indispensable si l'on résout des équations ou des systèmes d'équations par exemple.

2.3 Conversion de règles

Nous rencontrerons fréquemment le problème suivant : convertir une liste de règles en un ensemble de valeurs. Il est impératif de bien comprendre le fonctionnement des opérations qui suivent.
Résolvons l'équation $x^4 - 3\,x^3 - 2\,x^2 + 10\,x - 12 = 0$ et sélectionnons les racines réelles.
Nous ne nous attardons pas sur la syntaxe générale de `Solve`, assez intuitive dans cet exemple.

```
regles = Solve[x^4 - 3 x^3 - 2 x^2 + 10 x - 12 == 0 , x]
```

```
{{x → -2}, {x → 1 - i}, {x → 1 + i}, {x → 3}}
```

Comme prévu, on obtient une liste, de listes, de règles. Transformons en la liste, notée *ensemble*, des valeurs de x, solutions :

```
ensemble = x /. regles
```

```
{-2, 1 - i, 1 + i, 3}
```

Déterminons par exemple la sous-liste formée des racines réelles :

```
Cases[ensemble, t_ /; Im[t] == 0]
```

```
{-2, 3}
```

Traduction : recherchons dans *ensemble*, la sous-liste formée des variables t (le caractère de soulignement attaché à t lui confère le statut de variable, ce que nous détaillerons dans la notion de fonction), telles que la partie imaginaire de t soit nulle. C'est très proche de l'écriture mathématique d'un ensemble. Nous étudierons d'autres possibilités de *filtrage*, encore plus élégantes. Modifions la recherche :

```
Cases[ensemble, z_ /; Im[z] < 0 && Re[z] > 0]
```

```
{1 - i}
```

Recherchons maintenant les solutions complexes, non réelles.

```
Cases[ensemble, _Complex]
```

```
{1 - i, 1 + i}
```

Vous avez encore remarqué que pour Mathematica, un nombre réel n'est pas un nombre complexe! On peut dire qu'un nombre est de type `Complex` s'il s'écrit $a + i\,b$, avec $b \neq 0$.

Naturellement, nous pouvons évaluer une expression fonction de la variable x à partir des solutions de l'équation précédente. Nous obtiendrons une liste de quatre complexes, puisque l'équation possède quatre racines dans $\mathbb{C}$:

```
expr := (x^2 + x - 2)/(x^2 + i)
expr /. regles
```

$$\left\{0,\ 3 - i,\ 1 + \frac{i}{3},\ \frac{45}{41} - \frac{5\,i}{41}\right\}$$

3. Listes et expressions

3.1 Etude des listes

■ Généralités

La notion de liste (`List`), est un concept clé sous Mathematica, pour les raisons suivantes :
• Les primitives se référant aux listes sont nombreuses et optimisées.
• Les fonctions du système s'appliquent aux listes (elles possèdent l'attribut `Listable`) et impliquent une grande puissance.
• Il existe de nombreuses passerelles entre listes et expressions, concept unificateur qui englobe celui de liste, que nous étudierons après les listes.
• En programmation, une utilisation méthodique des listes et des primitives qui leur sont attachées, permettra de concilier trois avantages : puissance, concision, rapidité.

■ Notations

Dans ce qui suit, *l* désignera une liste, *expr* une expression, *f*, *g*, *h*, des fonctions.
Pour les primitives les plus explicites, seule la syntaxe générale est mentionnée : l'aide en ligne est remarquablement conçue et documentée, le lecteur y puisera de nombreux renseignements.