Chapitre III

SÉQUENCES, ENSEMBLES ET LISTES
FONCTIONS

Part I
Séquences, ensembles et listes

1 Séquences

Définition. On appelle **séquence** *(expression sequence)*[1] une collection ordonnée d'expressions séparées par des virgules, et éventuellement entre parenthèses.

Une séquence : *exp1*, *exp2*, . . . , *expn*

Exemples. *Effectuer :*

```
>3,4,5;
>s1:=a,b,c; s2:=(d,e);
>sequence:=s1,s2,f,g,h;
>seqvide:=NULL; # La séquence vide
>sequence[1],sequence[8];sequence[9];
>sequence[2]:=x; # Ceci est interdit
>sequence:=sequence[1],x,sequence[3..8];
```

Il faut bien comprendre que les séquences sont des structures en lecture seule. On ne peut pas modifier uniquement le kème terme de la séquence : il faut redéfinir entièrement la séquence.

Modes de définition des séquences. On a vu que l'on peut définir une séquence directement en écrivant une suite d'expressions séparées par des virgules. On peut aussi faire appel aux opérateurs `seq` et `$` de la façon suivante :
Effectuer :

```
>seq(2^i,i=1..10);
>2^i $ i=1..10;
>j $ 5;
>seq(k^2,k=1..n); # Ceci conduit à une erreur
>k^2 $ k=1..n; # C'est mieux ainsi
>subs(n=5,%);
>eval(%);
```

2 Ensembles

Définition. On appelle **ensemble** *(set)* une collection non ordonnée d'expressions toutes différentes séparées par des virgules, notées entre accolades

Un ensemble : {*exp1*, *exp2*, . . . , *expn*}

[1] *lit.* suite d'expressions

Exemples. *Effectuer :*

```
>{3,4,5};
>{5,4,3}; # Bien noter l'ordre
>{a,d,d,b,f,b,e,c,c};
>{seq(1/k,k=1..10)};
>ensvide:={}; # L'ensemble vide.
```

Y a-t-il un autre moyen de définir l'ensemble vide ?

Opérations sur les ensembles. On peut effectuer sur les ensembles les opérations de réunion (`union`), d'intersection (`intersect`), de différence ensembliste (`minus`).
Un ensemble est avant tout une expression, au même titre que `a*x+b`. On peut en extraire les opérandes (on obtient alors une séquence des éléments) ou le nombre des opérandes (on obtient le cardinal) à l'aide des fonctions `op` et `nops`.
Effectuer :

```
>ens1:={1,2,3,4,5};ens2:={seq(2*k,k=0..3)};
>"Réunion"=ens1 union ens2;
>Intersection:=ens1 intersect ens2;
>`Différence de ens1 et ens2`:=ens1 minus ens2;
>op(ens1);
>nops(ens2);
```

3 Listes

Définition. On appelle **liste** *(list)* une collection ordonnée d'expressions séparées par des virgules, notées entre crochets.

Une liste : `[`*exp1*`,`*exp2*`,...,`*expn*`]`

Opérations sur les listes. *Effectuer :*

```
>li1:=[Alain,Bernard,Céline];
>li1[3];
>li2:=[Daniel,Élise];
>seq3:=Frédérick,Guillaume;
>li3:=[seq3];
>`Mauvaise liste`:=li1,li2,li3;
>`Bonne liste`:=[op(li1),op(li2),op(li3)];
>`Pour la frime`:=[seq(op(li||k),k=1..3)];
>li1[2]:=Bertrand; # Cette méthode est à éviter
>li1:=subsop(2=Bruno,li1); # C'est la bonne méthode
(voir l'aide en ligne)
>li1;
```

Différence entre les trois structures. Une liste diffère d'un ensemble parce que ses éléments sont ordonnés et que la répétition y est autorisée.
Une liste diffère d'une séquence par le fait qu'elle n'est constituée que d'une seule expression, tandis qu'une séquence est la juxtaposition des expressions que sont ses opérandes. Ce point est important pour l'utilisation de certaines fonctions.
Effectuer :

```
>restart;
>subs(x=1,y=2,3*x+2,5*y+4);
>subs(x=1,y=2,[3*x+2,5*y+4]);
```

Plus tard. On parlera au moment de l'algèbre linéaire de la structure de tableau, qui permet la modification d'un des éléments sans avoir à redéfinir tous les éléments.

Part II
Fonctions

4 Définition des fonctions

4.1 Fonctions prédéfinies

Les fonctions usuelles sont prédéfinies. Ainsi, on peut utiliser `sin`, `exp`, `ln` etc. Nous avons déjà utilisé ces fonctions.

4.2 Opérateur flèche (->)

Pour définir une fonction, la première méthode consiste à utiliser l'opérateur flèche (noté par moins-strictement supérieur). La syntaxe est la suivante :

```
`nom de fonction`:=var(s)->expression en var(s)
```

Effectuer :

```
>restart
>f:=x->2*x^2+3;
>f(5);
>f(a+b);
>expand(%);
>g:=(x,y)->cos(x-y^2);
>g(Pi,sqrt(3*Pi)/2);
```

4.3 Fonctions définies par morceaux

On peut définir une fonction par morceaux à l'aide de l'opérateur flèche (`->`) et de la fonction `piecewise`. La syntaxe pour définir :

$$f\ :\ x \longmapsto \begin{cases} expr1 & \text{si } cond1 \\ expr2 & \text{si } cond2 \\ \vdots & \\ exprn & \text{sinon} \end{cases}$$

est :

```
f:=x->piecewise(cond1,expr1,cond2,expr2,...,exprn)
```

Effectuer :

```
>f:=x->piecewise(x<0,-x^2+1,x<1,-x+1,x^2-1);
>plot(f,-2..2);
```

4.4 Utilisation de `unapply`

C'est la fonction à utiliser pour transformer une expression en fonction.

```
f:=unapply(expression,variable)
```

Effectuer :

```
>expr:=3*x+4;
>f:=x->expr; # C'est un premier essai
>f(2);
>f:=unapply(expr,x); # C'est la bonne façon de faire
>f(2);
```

Remarque. L'expression qui suit l'opérateur flèche n'est pas évaluée au moment de la définition. Cela justifie les résultats qui suivent et qui montre l'un des intérêts de la fonction `unapply`.
Effectuer :

```
>restart;
>L:=[seq(x->x^k,k=1..4)];
>L[2]; # Ce n'est pas ce qui était attendu
>L1:=[seq(subs(n=k,x->x^n),k=1..4)]; # 1ère méthode qui marche
>L2:=[seq(unapply(x^k,x),k=1..4)]; # 2nde méthode qui marche
>L1[3](t);L2[3](t);
```

4.5 Utilisation des procédures

On étudiera plus en détail les procédures. Les fonctions, définies comme ci-dessus, sont de premiers exemples de procédures. L'utilisation des procédures permet de définir des fonctions d'une façon plus compliquée.
Effectuer :

```
>signe:=proc(x::realcons)
if x>0 then "+" elif x<0 then "-" else "0" fi
end;
> signe(3);signe(-1);
```

5 Opérations sur les fonctions

5.1 Composition

On peut composer deux fonctions à l'aide de l'opérateur . On peut composer une fonction `n`-fois avec elle-même en utilisant `n`.
Effectuer :

```
>g:=x->x+Pi/2;
>f:=cosg;
>simplify(f(x));
>h:=sin(g4);
>simplify(h(x));
```

5.2 Tracés de représentations graphiques

5.2.1 À partir d'une fonction

On utilise la fonction `plot` qui s'utilise en respectant la syntaxe suivante :

`plot(`*`fonction`* `, a..b , c..d , `*`options`*`)`

Les trois derniers arguments sont optionnels. Par défaut, l'intervalle horizontal `a..b` est fixé à `-10..10`, tandis que l'intervalle vertical *`c..d`* est calculé par Maple.

Attention ! Il faut que la fonction ne dépende d'aucun paramètre. Si la fonction que l'on veut représenter dépend d'un paramètre, on substitue à ce paramètre une valeur numérique à l'aide de `subs`.
Effectuer :

```
>plot(sin);
>plot(x->x^2*sin(1/x),-1..1);
>plot(t->3*a*ln(t),0..1);
```

`>H:=...` Définir la fonction $H : x \mapsto \dfrac{1}{1-x^2+\mathrm{i}\frac{x}{Q}}$

`>Hnum:=...` Définir la *fonction* `Hnum` où `Q` a pour valeur numérique 1

```
>G:=abs@Hnum; >plot(G,0..10);
>with(plots):
>semilogplot(20*log[10]@G,0.1..10,title="Diagramme de Bode");
```

5.2.2 À partir d'une expression

On utilise encore la fonction `plot`, mais en respectant la syntaxe suivante :

`plot(`*expression_en_*`x , x=a..b , c..d , `*options*`)`

Effectuer :

```
>plot(cos(x),x);
>P:=a*x^2+b*x+c;
>Pnum:=subs(a=1,b=-2,c=4,P);
>plot(Pnum,x=-5..5);
```

5.2.3 Les options de `plot`

On peut explorer les options suivantes (se référer à l'aide)

1. `discont=true` ou `discont=false` (défaut : `false`)
2. `numpoints=100` (défaut : 50)
3. `resolution=100` (défaut : 200)
4. `color=green` (défaut : red)
5. `title="Mon beau graphique"` (défaut : "")
6. `scaling=constrained`
7. On peut changer l'aspect du graphe après l'avoir tracé, en utilisant les boutons de la barre d'outils.

Tester les différentes options sur la représentation graphique de tangente.

5.2.4 Représentation simultanée de plusieurs graphes

Première méthode : Emploi d'un ensemble (ou d'une liste).
Effectuer :

```
>plot({cos(x),1-x^2/2},x=-Pi..Pi);
>approx:=seq(sum(x^k/k!,k=0..n),n=1..5);
>plot([exp(x),approx],x=-3..3,color=[red,blue $ 5]);
```

Deuxième méthode : Utilisation de `display`.
Chaque graphique est rangé dans une variable. La fonction `display` permet alors de présenter tous les graphiques en une seule fois :

display([*graphe1*,*graphe2*,...,*graphen*],*options*)

après avoir chargé le module `plots` et défini les variables ***graphe1*** etc.
Effectuer :

```
>restart;with(plots):
>g1:=plot(cos(x),x=-2*Pi..2*Pi,color=magenta):
>g2:=plot(sin(x),x=-2*Pi..2*Pi,color=blue):
>g3:=plot(tan(x),x=-2*Pi..2*Pi,-3..3,color=green,discont=true):
>display([g1,g2,g3],title="Principales fonctions trigonométriques");
```

5.2.5 Remarque : Tracés de lignes brisées

Effectuer :

```
>plot([[0,0],[1,3],[4,3],[0,0]]);
>plot([[0,0],[2,2],[0,2],[1,3],[2,2],[2,0],[0,2],[0,0],[2,0]],scaling=constrained,
    axes=none,color=aquamarine);
```

5.3 Utilisation de `map`

La fonction `map` permet d'appliquer une fonction à tous les opérandes d'une expression.
Effectuer :

```
>restart;
>f:=t->t^2
>map(f,a*x+b);
>map(f,t*(a+b));
>map(f,[seq(y||i,i=1..5)]);
```

Exercices

III.1 Dans $\mathbb{C}$, résoudre l'équation $z^2+\overline{z}+\mathrm{i}z=0$. Montrer que les points dont les affixes sont solutions forment un triangle rectangle que l'on représentera.

III.2 Déterminer les nombres complexes z tels que $\frac{z^2}{2z+3\mathrm{i}}$ soit imaginaire pur. Les représenter dans le plan complexe.

III.3

1. Montrer que :
$$f\ :\ z\mapsto\frac{z-2\mathrm{i}}{z+\mathrm{i}}$$
est une bijection de $\mathbb{C}\setminus\{-\mathrm{i}\}$ dans $\mathbb{C}\setminus\{1\}$.

2. On pose $z=x+\mathrm{i}y$ et $f(z)=u+\mathrm{i}v$, où $(x,y,u,v)\in\mathbb{R}^4$.
Exprimer x et y en fonction de u et v, et inversement.

3. Soit :
$$P=\left\{z\in\mathbb{C}\text{ t.q. }\mathrm{Im}(z)>\frac{1}{2}\right\}\qquad D=\{z\in\mathbb{C}\text{ t.q. }|z|<1\}$$

Démontrer que f induit une bijection de P sur D.

III.4 Soit f la fonction de $\mathbb{C}^*$ dans lui-même qui à z associe :

$$f(z) = \frac{1}{2}\left(z + \frac{1}{z}\right)$$

1. Déterminer et représenter l'image par f du cercle de centre O et de rayon R.
2. Déterminer et représenter l'image par f de la demi-droite d'origine O et d'angle polaire θ.