COLLES MAPLE N°1&2

Maple, comme vous le savez, contient plus de 2 500 fonctions pré-programmées. Évidemment, il est absolument impossible de les aborder toutes en classes préparatoires. Le but de nos TD sera plutôt, dans un premier temps, de mettre en pratique quelques notions de base sur la programmation en prenant pour sujets des fonctions mathématiques simples : nous fabriquerons ainsi « à la main », de nombreuses fonctions qui sont pourtant déjà pré-définies en langage Maple. Il s'agira en fait de découvrir, à travers des exercices concrets, la syntaxe Maple et des algorithmes de base. Ceci nous permettra, dans un deuxième temps, d'utiliser les formidables capacités de Maple en calcul formel pour travailler, sous un autre angle qu'en cours, les merveilleuses notions mathématiques que vous allez aborder avec madame Gornet. L'inconvénient de Maple, c'est que ce logiciel n'est pas libre : on ne peut pas savoir comment il marche....et il coute cher. Vous pouvez travailler chez vous sur deux logiciels gratuits : MuPAD et Xcas. Le deuxième, entièrement libre, a l'avantage de traduire les programmes Maple assez simples en langage TI89, ce qui peut intéresser certains d'entre vous...

Un de vos fidèles compagnons pendant ces TD sera le `?`. En effet, en cas de doute sur une fonction Maple, il suffira de taper

```
> ?lafonction;
```

pour voir apparaître une aide très complète avec de nombreux exemples d'utilisation [1]. Pour revenir à la feuille de calcul, un appui simultané sur *Ctrl* et $F4$ fera l'affaire.

Quelques conseils avant de travailler. Commencez chaque nouvel exercice par un

```
> restart;
```

pour « désaffecter » toutes les variables éventuellement utilisées dans un calcul précédent.
Si vous avez lancé un calcul qui a l'air de ne pas vouloir se terminer, cliquez sur l'icône STOP qui n'est active que lorsqu'un calcul est en cours.
Chacune de vos entrées débute après un *prompt* `>`, se termine par un `;` [2] ou un `:` [3] puis appuyez sur la touche *Entrée*. Si vous voulez passer à la ligne sans valider la ligne précédente, tapez $Shift$ + *Entrée*. Pour revenir plus haut, utilisez la souris.

Sept manières de calculer 7!

1) Bien sûr, on peut commencer par faire

   ```
   > 7!;
   ```

   mais le calcul de 7! n'est qu'un prétexte pour découvrir sur un exemple simple la syntaxe de programmation Maple.
2) On pourrait alors taper

   ```
   > 1*2*3*4*5*6*7;
   ```

   mais les choses pourraient se compliquer au moment de calculer 3232!
3) Nous allons donc utiliser une boucle for avec indice numérique : nous partons de 1, puis nous multiplions par 2, puis nous multiplions le produit précédent par 3, etc. Cela donne

   ```
   > p:=1: for k from 1 to 7 by 1 do p:=p*k od;
   ```

   Ce programme n'est pas optimum. En fait, dans une boucle for, les instructions $from$ et by sont facultatives : elles valent par défaut 1. Ici, on peut se contenter de

   ```
   > p:=1 for k to 7 do p:=p*k od;
   ```

   Il reste un inconvénient : Maple affiche tous les résultats intermédiaires. Rappelez-vous des rôles de `:` et `;` puis écrivez un programme qui n'affichera que la valeur finale.

1. Vous découvrirez en même temps pourquoi vous avez été en cours d'anglais depuis tant d'années.
2. si vous voulez que le résultat soit affiché.
3. si vous voulez que Maple exécute sans afficher.

4) Nous aurions pu créer la liste des entiers de 1 à 7 puis utiliser une boucle *for* indexée cette fois par les éléments de la liste. C'est plus compliqué, mais cela nous permet de découvrir comment Maple traite les listes. Il faudra bien distinguer

▷ les **ensembles** (*set*) qui sont des collections *non ordonnées* d'expressions toutes différentes séparées par des virgules et encadrées d'accolades.

```
> ens1:={2,4,1}; ens2:={2,5,8,5};
```

On peut effectuer les opérations usuelles sur les ensembles

```
> ens1 intersect ens2; ens1 union ens2; ens1 minus ens2;
```

l'ensemble vide se note

```
> { }
```

▷ les **suites** (*sequence*) qui sont des collections *ordonnées* d'expressions, différentes ou non, séparées par des virgules et encadrées ou non par des parenthèses.

```
> 5,7,5,1,2,3;
```

On peut aussi utiliser les opérateurs `seq` et `$` pour des suites définies par une formule explicite

```
> seq(k**2,k=1..5);

> p**2 $ p=1..5;

> m $ 5;

> seq(i**2,i=1..n); # problème...#

> j**2 $ j=1..n; # plus de problème #
```

On peut changer un élément de la suite

```
> subs(n=5,"); #remplacez " par % dans les versions ultérieures à Maple V.4#

> eval(");
```

▷ les **listes** (*list*) qui sont des collections *ordonnées* d'expressions séparées par des virgules et encadrées par des crochets. La différence, c'est qu'une suite, en tant que juxtaposition d'expressions, est en quelque sorte « en lecture seule », alors qu'une liste est une expression en elle-même et pourra donc « subir » des opérations algébriques.
Notez au passage quelques fonctions utiles

```
> s1:=i $ i=-2..2; s2:=a,b,c,d,e;

> l1:=[s1]; l2:=[s2]; # une liste est une suite entre crochets#

> nops(l2); # nombre d'opérandes #

> l2[3]; op(3,l2); # extrait le 3ème opérande #

> l2[3..5]; op(3..5,l2);

> select (x->x>0,l1); remove(x->x<0,l1);

> subs(b=32,l2); subsop(2=z,l1); subsop(5=NULL,l2); # pour substituer ou supprimer un
opérande #

> map(cos,l2);

> zip((x,y)->x+y,l1,l2);
```

Cela donne

```
> restart;

> l:=[k $ k=1..7]: p:=1: for i to 7 do p:=p*l[i] od: p;
```

5) Nous pouvons utiliser une boucle *while*

```
> restart;
```

```
> p:=1: k:=1: while k<7 do k:=k+1: p:=p*k od: p;
```

6) C'est bien beau, mais que ferons-nous quand il faudra calculer 32! ou 3232! et tous les autres ? Il faudrait créer un programme (on dira une **procédure**) qui donne $n!$ pour tout entier naturel n.

```
> restart; # je ne vous le dirai plus #
```

```
> fact:=proc(n)
local p,k; # nous aurons besoin de variables locales i.e. internes à la procédure #
p:=1;
for k to n do p:=p*k od;
end: # termine la procédure #
```

```
> fact(32);
```

7) Le meilleur pour la fin : la **procédure récursive**, qui s'appelle elle-même

```
> factr:=proc(n)
if n=0 then 1 # une boucle if...then...else pour régler le cas de 0! #
else n*factr(n-1)
fi; # symbolise la fin de la boucle #
end:
```

```
> factr(32);
```

En fait, ce mécanisme correspond à une suite numérique qui s'écrirait mathématiquement $u_n = n \times u_{n-1}$.

Exercice 1 Partie entière
Déterminez une procédure `E:=proc(x)` qui, à un réel positif x, associe sa partie entière.

Exercice 2 Valeur absolue
Déterminez une procédure `ab:=proc(x)` qui, à un réel x, associe sa valeur absolue.

Exercice 3 Moyenne
Déterminez une procédure permettant de calculer la moyenne des éléments d'une famille de nombres réels.

Exercice 4 Suite

Pour $n \in \mathbb{N}^*$, on considère $a_n = 1 + 1/n$ et $b_n = a_n^{a_n^{\cdot^{\cdot^{\cdot^{a_n}}}}}$, avec « n fois a_n ».
Par exemple, $b_2 = (3/2)^{3/2}$.

1) Déterminez une procédure qui calcule b_n à partir de n.
2) On admet que la suite (b_n) est décroissante et converge vers 1. Déterminez une instruction qui permette de déterminer le plus petit entier n tel que $b_n \leqslant 1{,}001$.

Exercice 5 Somme
Vous savez peut-être que la suite $(S_n)_{n\in\mathbb{N}}$ de terme général

$$S_n = \sum_{k=0}^{n} \frac{1}{k!}$$

est croissante et converge vers e. Nous l'admettrons dans cet exercice.

1) Déterminez une procédure `S=proc(n)` qui, à un entier naturel n, associe S_n. N'oubliez pas les gages habituels : vous n'utiliserez ni la fonction prédéfinie `sum`, ni les procédures calulant $n!$ vues précédemment.
2) Déterminez une procédure `seuil:=proc(p)` qui, à un entier naturel p, associe le plus petit entier naturel n tel que $|S_n - e| \leqslant 10^{-p}$. Vous aurez besoin de savoir que e se dit `exp(1)` en Maple.

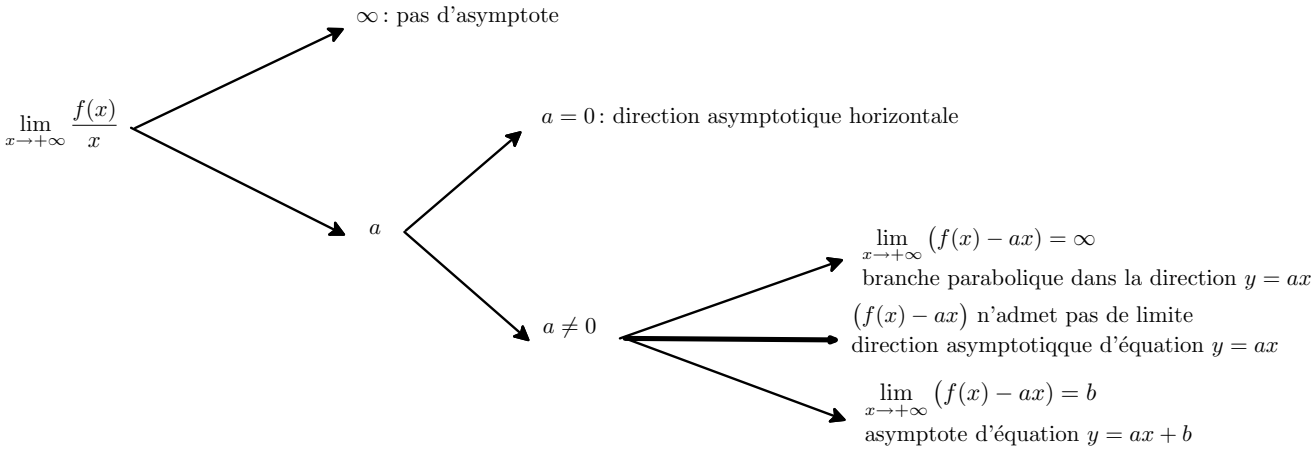
Exercice 6 Équation du second degré
Déterminez une procédure `sol:=proc(a,b,c)` qui, à une équation $ax^2 + bx + c = 0$, associe son ensemble des solutions.

Exercice 7 test sur une liste
Déterminez une procédure `test:=proc(l)`, `l` étant une liste d'entiers, qui teste si ses éléments forment une suite croissante.

Exercice 8 Coefficients du binôme
Vous connaissez tous la formule du triangle de Pascal. Déterminez donc une procédure récursive `c:=proc(n,p)` qui calcule les coefficients du binôme. Testez avec `c(32,3)` et `c(3,32)`.

Les objets Maple - Étude des branches infinies

Pour vous, x est un réel, n un entier, [1,2] un intervalle, etc. pour Maple, cela n'a rien d'évident : il faut parfois tout lui expliquer.
Pour illustrer notre propos, nous allons écrire un programme étudiant les branches infinies d'une fonction numérique. Voici un bref rappel de cours :



Il va falloir utiliser pas mal de boucles if qui correspondent à chaque embranchement de l'arbre. Il va d'abord falloir demander à Maple de calculer des limites. Il sait faire :

```
> limit(sin(x)/x,x=0); limit(1/sqrt(x),x=0); limit(sin(x)/x,x=infinity); limit(sin(x),x=infinity);
```

Maintenant, il va falloir tester la réponse :a est réel ou infini ? La limite existe ou n'existe pas ? Pour cela, commençons par explorer la fonction `whattype(expression)`

```
> whattype(32); whattype(1/2); whattype(x->2*x); whattype(limit(sin(x),x=0));
```

```
>  whattype(limit(32*x,x=infinity)); whattype(limit(sin(x),x=infinity));
```

Ensuite, il existe une fonction qui teste si une expression est d'un type déterminé :`type(expression,domaine)`.

```
> type(32,integer); type(32.1,integer); type(limit(sin(x),x=infinity),infinity);
```

Nous allons maintenant construire une procédure `branche:=proc(f)` Qui renvoie l'éventuelle branche infinie de la fonction f. Elle contiendra deux variables locales a et b. Je vous livre le début

```
> branche:=proc(f)
```

```
> local a,b;
```

```
> a:=limit(f(x)/x,x=infinity):
```

```
> if type(a,..)=true then lprint(‘pas d’asymptote‘);
```

```
> elif type(a,infinity)=true then lprint(‘pas d’asymptote‘);
```

À vous d'imaginer la suite... Pour demander à Maple de envoyer un message mélangeant du texte et un résultat de calcul, on peut utiliser la commande `printf(‘asymptote d’equation y=%f x+%f‘,a,b);` où l'objet flottant `f` (c'est à dire le réel `f`) va prendre les valeurs de a et b.