Maple TD1

1 Rappels

1.1 Obtenir de l'aide

Les fonctionnalités offertes par Maple sont trop nombreuses pour pouvoir espérer retenir la syntaxe de chacune des commandes. Aussi est-il essentiel d'avoir accès à la liste de toutes les commandes disponibles, ainsi qu'au descriptif détaillé de chaque commande.

Essayez :

```
>?
>?exp
```

1.2 fonctions usuelles, constantes

Maple est une sorte de calculatrice programmable très sophistiquée. Les opérations élémentaires (+,-,*,/) sont donc évidemment prises en charge, ainsi que les fonctions usuelles (**sin**, **cos**, **exp**, **arcsin**, **ln**, **log10**, **sqrt** ...) et des constantes numériques (**Pi**, **Gamma**, **I**...)

1.3 Types de données

En mathématique, on distingue, de manière souvent implicite, plusieurs types d'objets : les ensembles, les fonctions, les réels, les complexes, les polynômes... Maple, comme la plupart des langages de programmation, reconnaît aussi plusieurs types de données. Il différencie les entiers (**integer**), les "nombres à virgule flottante" (**float**), les procédures (**procedure**), les listes (**list**), etc. Pour avoir le type d'une expression, on utilise la commande **whattype**.

Essayez :

```
>whattype(12);  >whattype(12.45);  >whattype(Pi);  >whattype(PI);
 >f:=x->x^2;  >whattype(f)  >g := proc(x);x^2;end: whattype(g);
```

1.4 Calcul symbolique et évaluation numérique

Par défaut, Maple cherchera toujours à renvoyer des valeurs exactes, quitte à n'en donner qu'une valeur littérale. Les instructions **evalf**, **evalb**, **evali** permettent de forcer l'évaluation numérique approchée du résultat, suivant le type que l'on veut.

Essayez :

```
>cos(2*Pi/7);  >evalf(cos(2*Pi/7));  >evalb(2 = 1*2);
```

1.5 Quelques outils mathématiques plus élaborés

Maple fourmille de commandes spécialisées et d'outils de calcul extrêmement utiles. Voici un rapide aperçu de quelques-uns des plus simples d'entre eux. N'hésitez pas à consulter l'aide pour plus de détails !

- résolution d'équations : **solve ;**, extraction de racines : **roots ;**
- intégration : (int ;) et dérivation : (diff ;)
- calcul de limites : (limit ;)
- factorisation : (factor ;), développement : (expand ;) et simplification : (simplify ;)

Souvent, ces commandes peuvent aussi s'écrire avec une majuscule au début, essentiellement pour de l'affichage.

```
>Int(sin(x),x=0..Pi) = int(sin(x),x=0..Pi);
```

1.6 Représentation graphique

On peut tracer le graphe d'une fonction sur un ensemble de points :

```
 > plot(sin(x)/x,x=0..2*Pi);
```

1.7 Déclaration et utilisation de variables

En Maple, comme dans tout langage de programmation, il est possible d'attribuer des valeurs à des variables : cela permet de mémoriser le résultat d'un calcul en lui donnant un nom, puis utiliser ce nom chaque fois que nécessaire dans les calculs suivant :

```
> x1 := sin(2*Pi/7);  > x2 := cos(2*Pi/7);  > evalf(x1^2+x2^2);
```

Quelle différence y a-t-il entre **x1=12 ;** et (x1 :=12 ;) ?

1.8 Procédures

Les résultats que l'on cherche à établir à l'aide de Maple ne sont pas forcément obtenus en une seule ligne de commandes : ils sont bien souvent le fruit d'une longue série d'instructions qu'il faut regrouper une fois pour tout sous un simple nom, afin de pouvoir effectuer la tâche autant de fois que l'on veut, comme s'il s'agissait d'une commande Maple prédéfinie. Une telle séquence automatisée d'instructions s'appelle une procédure : elle dépend souvent d'un certain nombre d'arguments d'entrée, et peut donc être vue comme une fonction. La procédure qui suit, par exemple, automatise le calcul de la somme des n premiers entiers :

```
 > Somme := proc(n);
     sum(k,k=1..n);
   end;
```

Vous pouvez terminer les instructions par **:** au lieu de **;** si vous ne voulez pas afficher le résultat.

2 Exercices

Sauvegardez souvent, certains exemples pouvant parfois planter Maple. Utiliser la commande **restart** avant de commencer.

2.1 Factorielle

Comme tout langage de programmation, Maple offre la possibilité d'écrire des boucles for. Par exemple, la procédure qui suit imprime à l'écran les n premiers entiers strictement positifs :

```
imprime := proc(n);for i from 1 to n do print(i) end do; end;
```

1. Comment marche une boucle for ?
2. Ecrire une procédure calculant $n!$ à l'aide d'une boucle for.
3. En fait, $n!$ est définie mathématiquement de manière récursive : $n! = 1$ si $n = 1$, $n(n-1)!$ si $n \geq 2$. $n!$ se calcule donc de proche en proche, à partir des valeurs précédentes. Maple permet de coller à ce modèle mathématique puisqu'une fonction peut s'appeler elle même. Vous pouvez tester la valeur d'une variable et effectuer des actions en fonction du résultat à l'aide des commandes **if**, **then**, **elif** et **else**. Définir une fonction factorielle récursive. Que se passe-t-il si vous ne faites pas le test $if(n = 1)$?
4. Calculer le nombre de 0 à la fin de 42!. (vous pourrez utiliser la fonction **ifactor**).

2.2 Suite de Fibonacci

Rappelons la définition de la suite de Fibonacci :

$$\begin{cases} u_0 = 1 \\ u_1 = 1 \\ u_{n+2} = u_n + u_{n+1} \end{cases}$$

1. Ecrire une procédure fib qui calcule récursivement u_n
2. Essayer fib(5) puis fib(25). Quel est le problème lorsque n devient trop grand ?
3. On rappelle que % et %% renvoient aux résultats précédents calculés par Maple. Ces fonctions sont très pratiques, mais il ne faut pas en abuser, car elles supposent que vous effectuez toutes vos commandes dans l'ordre où elles sont écrites. Calculer fib(5) avec % et %%
4. Ces suites sont en fait des cas particuliers des suites de la forme $u_n = au_{n-1} + bu_{n-2}$. Rappeler sur un papier comment étudier ces suites.
5. Ecrire une procédure prenant comme paramètre u_0, u_1, a et b et retournant le terme général.
6. On note $T(n)$ le nombre de fois que la fonction fib est appelée lorsque l'on appelle $fib(n)$. Trouver une relation de récurrence sur $T(n)$. En déduire $T(n)$ pour tout n.

La croissance extrêmement rapide de la suite $T(n)$ explique le problème rencontré en 2). Finir le calcul suppose de faire une quantité trop importante d'appels, le rendant impossible, on parle de complexité temporelle.

2.3 Suite de Syracuse

Par définition, la suite de Syracuse est :

$$\begin{cases} u_0 \text{un entier strictement positif} \\ u_{n+1} = 3u_n + 1 \text{ si } u_n \text{ est impair} \\ u_{n+1} = \frac{u_n}{2} \text{ si } u_n \text{ est pair} \end{cases}$$

1. Ecrire une fonction f prenant en argument u_n et renvoyant u_{n+1}
2. Tester la fonction f en calculant les 4 premiers termes de la suite définie par $u_0 = 4$.
3. Que se passe-t-il si à un moment $u_n = 1$?
4. On appelle orbite d'un entier positif u_0 la séquence formée par $u_0, u_1, ..., u_n$ avec n le premier indice pour lequel $u_n = 1$. Ecrire une procédure prenant u_0 et donnant son orbite.
5. Il n'est pas évident que cette fonction termine, c'est en fait encore un problème ouvert. Ecrire une procédure qui prend en paramètre n et vérifie pour $u_0 \in [1..n]$ calcule la taille maximale des orbites (utilisez la fonction *nops*).
6. Ecrire une procédure effectuant le tracé d'un orbite, c'est à dire la ligne joignant les points (k, u_k). Utiliser les fonctions plot et seq.
7. Ecrire une procédure effectuant le tracé d'un orbite en joignant les points (u_0, u_0), (u_0, u_1), (u_1, u_1), (u_1, u_2)... jusqu'à $u_n = 1$. Vous pourrez utiliser la fonction map. Ajoutez au dessin les droites $y = x$, $y = \frac{x}{2}$ et $y = 3x + 1$.

2.4 Suites $u_n = f(u_{n-1})$

1. Ecrire une procédure **escargot** qui trace les termes de la suite récurrente $u_{n+1} = f(u_n)$. Les arguments seront n le nombre de terme, f la fonction, et u_0. Tracer aussi $y = x$ et $y = f(x)$ sur le même dessin.
2. Appliquer la procédure à la fonction $\frac{x+\frac{a}{x}}{2}$. C'est l'algorithme de Babylone qui permet de calculer une valeur approchée de $\sqrt{a}$.
3. Appliquer la procédure à la fonction $ax(1-x)$ pour $0 < a < 4$. Essayer différentes valeurs de a puis faites tendre a vers 4. C'est la suite de Feigenbaum, modélisant l'évolution d'une population.
4. Comment analyser ces suites en général ? Comment Maple peut aider aux différentes étapes ou donner le résultat final ?

2.5 Triangle de Sierpiński

Proposez, sur une feuille, un algorithme pour tracer cette suite de dessins :

C'est un dessin fractal, appelé triangle de Sierpiński. Implémentez une procédure donnant le triangle à l'itération n.