TD Maple 1 : Un peu de Maple et d'algorithmie

Michael Monerau

30 septembre 2010

Introduction

L'objectif de cette année de TDs est de maîtriser Maple assez pour que ça devienne un outil agréable de travail, où vous sachiez résoudre / implémenter rapidement un calcul formel ou un algorithme que vous avez en tête.

Quelques conseils en vrac :

- Retrouvez les TDs et corrigés sur http ://www.eleves.ens.fr/home/monerau
- Utiliser l'aide `:?commande` affiche l'aide sur "`commande`". Ca ne sert à rien d'apprendre les ordres d'arguments par coeur, etc. Il faut retenir le sens des fonctions, qu'elles existent. Les détails de syntaxe se retrouvent dans l'aide, ne vous encombrez pas l'esprit pour rien.
- Vous pouvez m'envoyer vos fichiers par mail si vous voulez que je relise votre code (adresse mail sur ma page)
- Ecrivez toujours des commentaires sur ce que vous faites : hypothèses sur les arguments, valeur de retour
- Quand vous écrivez une fonction, testez la systématiquement
- Essayez de séparer vos fonctions en un maximum d'opérations logiques différentes (et donc de fonctions différentes), ça limite les erreurs et permet plus de ré-utilisation
- Ne faites jamais de copier-coller pour utiliser des résultats. Mettez les plutôt dans des variables intermédiaires. C'est plus propre, et si vous changez le calcul au début, le changement sera automatiquement retranscrit dans toute la suite
- Indentez votre code de façon claire pour le rendre facilement lisible

1 Les éléments de base

1.1 Assignation

Pour assigner un objet à une variable, il faut utiliser `:=`. Pour "libérer" une variable, utiliser `unassign`. Par exemple :

```
arbre := 'hou'
unassign('arbre');
```

1.2 Sequences / Lists / Sets

Sequences : suite d'éléments. S'écrit sous la forme `1,2,3,4` par exemple. On accède aux éléments par `l[2]`. Les indices commencent à 1 et non à 0. On récupère la longueur de la séquence par `nops(s)`.

On ne peut pas modifier les éléments une fois que la liste est créée.

La séquence vide s'écrit `NULL`.

On concatène des séquences avec `s1, s2`.

Lists : séquence entre crochets. Comme les sequences mais les éléments sont modifiables.

Sets : séquence entre accolades. Non ordonné, pas de duplicata.

On passe d'une séquence à une liste en mettant des crochets autour. On passe d'une séquence à un set en mettant des accolades autour.

`op` récupère une séquence sous-jacente à un set ou à une liste.

Question 1. Sequence

Créer la séquence des monômes x^i pour i de 0 à 50. En faire une liste. Remplacer x^{25} par x^{12}. Combien la liste a-t-elle d'éléments ? En faire un set. Combien le set a-t-il d'éléments ?

Question 2. Aplatir

Écrire une fonction qui transforme une liste de listes en entrée en l'ensemble des éléments de ces listes.

1.3 Fonctions / Expressions

Il y a une différence entre l'expression `expr := 3*x^ 2 + 5*x + 3` et la fonction `f := x -> 3*x^ 2 + 5*x + 3`.

La première s'évalue en utilisant la fonction `subs(x = 2, expr)` alors que la fonction s'évalue simplement avec `f(2)`.

La première est souvent plus facile à manipuler au travers des algorithmes. Cependant, elle nécessite une certaine discipline sur les noms des arguments (par exemple il faut savoir que l'argument se nomme `x` pour pouvoir évaluer `expr`, ça peut être problématique quand on passe une fonction en argument). La deuxième est plus pratique à évaluer, il n'y a pas à se soucier du nommage des variables. Mais la manipulation est plus difficile. On peut retrouver l'expression en évaluant `f(x)` avec `x` non assignée.

Question 3. Conversion

Utiliser la fonction `unapply` pour passer de l'expression `3*x^ 2 + 5*x + 3` à la fonction polynomiale associée. Repasser alors de la fonction à l'expression.

Évaluer le polynôme en 2 en utilisant l'expression et la fonction.

1.4 Afficher les itérations de suites récurrentes

Le but est ici de faire une procédure pour tracer les diagrammes en escargot pour les suites récurrentes $u_{n+1} = f(u_n)$. On pourra conserver ce code pour le réutiliser au cours de l'année.

Question 4. Escargot

Écrire une procédure qui prend en arguments une expression f (ou une fonction, comme vous voulez), la valeur u_0 de début de l'itération et le nombre de pas à dessiner n_{max}.

On pourra utiliser les fonctions `plot` (observer ce qu'il se passe quand on passe une liste ou un set), son option `discont` pour améliorer le rendu ; et la fonction `display`. Il faut importer le package `plots` avec `with(plots):`.

(a) Étudier le cas où $f(x) = x^2 - 1$.

(b) Étudier le cas où $f(x) = \frac{1}{\ln x}$. Cette suite est-elle bien définie ?

(c) **Suite logistique.** Étudier le cas où $f(x) = \mu x(1-x)$, avec μ variant entre 0 et 4.

2 Récursivité

2.1 Coefficients binômiaux

Question 5. Écrire une procédure récursive qui donne le coefficient binomial C_n^k.

Question 6. En déduire une procédure qui développe $(a+b)^n$. L'utiliser pour développer $(a+b)^{50}$ avec a et b non assignés. *(Remarquer la différence entre* `add` *et* `sum`*)*

2.2 Les tours de Hanoï

Le problème des tours de Hanoï, c'est des moines qui bougent des disques de rayons tous différents sur trois poteaux. Il y a deux règles :

- Les disques ne peuvent être bougés qu'un à un
- Un disque A ne peut être posé sur un autre disque B que si le rayon de A est plus petit que celui de B.

On commence le jeu avec n disques posés les uns sur les autres (en pyramide donc) sur le poteau de gauche. Le but est de passer la pyramide sur le poteau de droite.

Question 7. Tours de Hanoï

Écrire une procédure récursive qui donne les mouvements à réaliser pour y arriver (numéro du disque à bouger, et mouvement à faire).

2.3 La suite de Fibonacci

La suite de Fibonacci est définie par $\mathcal{F}_0 = 0$, $\mathcal{F}_1 = 1$, $\forall n \geq 0, \mathcal{F}_{n+2} = \mathcal{F}_n + \mathcal{F}_{n+1}$.

Question 8. Fibonacci

Écrire une procédure récursive qui prend en argument n et qui renvoie $\mathcal{F}_n$. Demander des valeurs pour n grand. Si c'est trop long, trouver comment réduire le temps.

3 Rendez la monnaie

Il s'agit ici d'écrire un algorithme de rendu de monnaie d'une machine à café.

En entrée de la procédure, on dispose de deux listes : une qui contient les valeurs des pièces disponibles, et la deuxième qui donne le nombre de pièces disponibles pour chaque type.

Par exemple, $[1, 0.5, 0.2, 0.1]$ et $[8, 10, 2, 1]$ veulent dire qu'il y a 8 pièces de 1€, 10 pièces de 50 centimes, etc.

En plus de ça, on reçoit le montant à payer et le montant inséré dans le machine.

Question 9. Et avec ceci ?

Ecrire une procédure qui calcule une liste des pièces à rendre (sur le modèle de la deuxième liste passée en argument).

Indication : On pourra utiliser un algorithme glouton : on essaie de rendre un maximum de pièces de valeur maximale. Quand on ne peut plus en rendre sans dépasser, on essaie de rendre un maximum de pièces de la valeur juste inférieure, et ainsi de suite jusqu'au bout.

Question 10. Faire mieux ?

L'algorithme glouton est-il optimal pour le rendu de monnaie ?