Quelques commandes XCAS

Ce document ne peut évidemment faire le tour des possibilités du logiciel XCAS. Son objectif est d'apporter quelques éléments utiles à la réalisation des exemples demandés. Une aide, très fournie, est accessible via le bouton `?`.

XCAS et la programmation

XCAS permet d'écrire des programmes, comme n'importe quel langage de programmation. Voici quelques caractéristiques :

1. En utilisant la combinaison de touches `alt`+ `p`, la saisie d'un programme peut-être rendue plus confortable.
2. Les caractères // permettent d'insérer des commentaires.
3. **Ne pas confondre :**
   - le signe := qui désigne l'affectation
   - le signe == qui désigne une égalité booléenne : c'est une opération binaire qui retourne 1 pour `Vrai` ou 0 pour `Faux`)
   - le signe = utilisé pour définir une équation.
4. Il n'y a pas de distinction entre programme et fonction :
   une fonction renvoie la valeur de la dernière instruction évaluée ou ce qui suit le mot réservé `return`.
   ☞ *Comme pour tous les environnements de calcul, programmer consiste à étendre XCAS en lui rajoutant les fonctions souhaitées. Structurer la programmation consiste à hiérarchiser les différentes fonctions qui s'appellent entre elles.*
5. Le langage est non typé. On distingue seulement les variables globales, qui ne sont pas déclarées, et les variables locales, déclarées en début de fonction par l'intermédiaire de l'instruction `local`.
   ☞ *Même si le langage est non typé, il est donc recommandé d'initialiser les variables avant de les utiliser.*
6. Certains mots sont réservés au calcul formel, et ne peuvent être utilisés comme nom de variables.
   C'est le cas, par exemple de `i`, `e` et `pi`.
   ☞ *on veillera, en particulier, à ne pas utiliser la lettre* i *comme compteur de boucle.*
7. La syntaxe de déclaration d'une fonction est la suivante.

```
nom_fonction(var1,var2,...):=
{
local var_loc1, var_loc2,... ;
instruction1;
instructions ...}
```

8. La syntaxe des tests est celle-ci :

```
if (condition) {clause_vraie} else {clause_fausse}}
```

(le `else` est facultatif). La condition est un booléen, résultat d'une expression logique, utilisant les opérateurs habituels.
☞ *On peut aussi utiliser la syntaxe*
«si <condition> alors <clause vraie> sinon <clause fausse> fsi;»

9. La syntaxe d'une boucle `pour` est la suivante.

   ```
   for(initialisation;test;incrementation){ corps_de_boucle }
   ```

   ☞ *On peut aussi utiliser la syntaxe*
   «pour <variable> de <début> jusque <final> pas <incrementation> faire <corps de boucle> fpour;»
10. La syntaxe d'une boucle `Tantque` est la suivante.

    ```
    while (condition) { corps_de_boucle }
    ```

    ☞ *On peut aussi utiliser la syntaxe*
    «tantque <condition> faire <corps de boucle> ftantque;»

Un exemple de programme

```
u(n):={ //déclaration de la fonction dont le paramètre est noté n
local k,fact; // déclaration des variables locales
fact:=1; // initalisation d'une variable
pour k de 1 jusque n faire  // boucle
                     fact:=fact*k
fpour;
return fact;} // on renvoie la valeur fact = n!
```

XCAS un logiciel de calcul formel

Ce paragraphe est fortement «inspiré» du tutoriel :
« Démarrer en calcul formel » de R. De Graeve, B. Parisse, B. Ycart.
Il ne saurait en aucun cas se substituer à une lecture de ce dernier disponible ici : http ://www-fourier.ujf-grenoble.fr/ parisse/giac/doc/fr/tutoriel/index.html

1. On peut faire certains types d'hypothèses sur une variable avec la commande `assume`, par exemple `assume(a>2)`.
2. **Réécriture d'expressions**
   - `expand` : développe une expression en tenant compte uniquement de la distributivité de la multiplication sur l'addition et du développement des puissances entières.
   - `normal` et `ratnormal` écrivent une fraction rationnelle (rapport de deux polynômes) sous forme de fraction irréductible développée ; `normal` tient compte des nombres algébriques (par exemple comme `sqrt(2)`) mais pas `ratnormal`.
   - `factor` elle écrit une fraction sous forme irréductible factorisée.
3. **Fonctions**
   - Pour créer une nouvelle fonction, il faut la déclarer à l'aide d'une expression contenant la variable.
     Par exemple l'expression x^2-1 est définie par xˆ2-1. Pour la transformer en la fonction f qui à x associe x^2-1, trois possibilités existent :

```
f(x):= x^2-1
f:=x->x^2-1
f:=unapply(x^2-1,x)
```

- La fonction `diff` permet de calculer la dérivée d'une expression par rapport à une ou plusieurs de ses variables.
  Pour dériver une fonction f , on peut appliquer `diff` à l'expression f (x), mais alors le résultat est une expression.
  Si on souhaite définir la fonction dérivée, il faut utiliser `function_diff`.

```
E:=x^2-1
diff(E)
f:=unapply(E,x)
diff(f(x))
f1:=function_diff(f)
```

- XCAS étant un langage fonctionnel, l'argument d'une fonction peut être une autre fonction. Si c'est le cas, on peut soit donner le nom de la fonction argument dans la commande, soit sa définition.
  Par exemple `function_diff(f)` ou bien `function_diff(x->x^ 2)`.
- Pour afficher une courbe, on utilise l'instruction `plot` avec en paramètres une expression ou une liste d'expressions dont on veut la représentation graphique, puis la variable (éventuellement on indique l'intervalle de valeurs de la variable).

4. XCAS distingue plusieurs sortes de **collections d'objets**, séparés par des virgules :
   - les listes (entre crochets)
   - les séquences (entre parenthèses)
   - les ensembles (entre pourcentage-accolades)

   La séquence vide est notée `NULL`, la liste vide `[ ]`.

5. **Objets graphiques**
   L'objet graphique est affiché dans la fenêtre `DispG` (Display Graphics) que l'on fait apparaître par le menu du bandeau supérieur
   Session->Show->Show DispG ou avec la commande `DispG()`
   ☞ *les graphiques successifs sont alors tracés individuellement dans chaque fenêtre de réponse, et superposés dans la fenêtre `DispG`.*
   On peut effacer la fenêtre `DispG` par la commande `erase()`.