TP 1 : Débuter avec Xcas

Xcas est l'un des trois logiciels de calcul formel disponibles pour l'oral de modélisation d'option C, avec Maxima et Sage. Cette première séance permettra de se familiariser avec ses fonctionnalités de base et ses usages possibles.

1 Installer, lancer et découvrir l'interface de Xcas

1.1 Installation et lancement

Le site de référence pour les versions de Xcas (chaque OS a la sienne) est

http://www-fourier.ujf-grenoble.fr/~parisse/install_fr.html

La page parente est d'ailleurs la page à consulter pour toute aide ou information complémentaire :

http://www-fourier.ujf-grenoble.fr/~parisse/

Après l'installation, lancer Xcas dépend de l'OS et de l'ordinateur utilisés. En salle de TP, il suffit de lancer un terminal (menu Système ou Accessoires) puis d'entrer `xcas` ou bien de lancer depuis le menu éducation (le jour J, il y aura une icône pour lancer le logiciel).

Il existe également une version de Xcas en ligne accessible depuis Firefox par exemple, à l'adresse

http://www-fourier.ujf-grenoble.fr/~parisse/xcasfr.html

1.2 Configuration du CAS

La première chose à identifier est la zone cliquable

`Config : exact real RAD 12 xcas` (etc)

(ou bien aller dans `Cfg->Configuration du CAS`). Elle dit sous quelle configuration tourne Xcas au moment de l'usage. Voici la configuration par défaut suggérée pour débuter :

- `Progstyle` : `Xcas`
- `radian` coché, les autres cases décochées, pour du calcul exact dans le corps des coefficients.

1.3 Interface et aide

L'aide complète pour débuter se trouve dans le logiciel lui-même et *sera donc accessible le jour J* : y aller via `Aide->Débuter en calcul formel->Tutoriel`.

Ensuite, l'interface se compose des éléments suivants au démarrage, de haut en bas :

- Barre de menu (de `Fich` à `Tortue`)

— En jaune juste en-dessous : onglet indiquant la session.
— `?` pour l'index des commandes, `Sauver` pour enregistrer les commandes, la configuration du CAS, `STOP` pour interrompre un calcul, `Kbd` (Keyboard) pour faire apparaître un clavier, `X` pour fermer la session (utiliser `Fich->Nouvelle session` pour en relancer une)
— Finalement, une ligne de commande, dans laquelle on peut taper des instructions puis `Entrée` pour un résultat : tester `2+2`.

1.4 Commandes les plus utiles de l'interface

Les commandes les plus utiles de l'interface (avec les raccourcis équivalents) sont sans doute les suivantes :

— `Aide->Index` : index cherchable des commandes Xcas avec description brève. Également accessible avec `F1` après ou avant une (partie de) commande tapée, et `?` en préfixe d'une commande connue.
— `Aide->Manuels->Algorithmes` fournit un manuel qui contient au moins tout le programme d'option C en local (donc aussi accessible le jour J).
— `Outils` contient des listes de commandes de base si on a oublié comment les écrire. En particulier, `Alt+N` ouvre une nouvelle ligne de commande et `Alt+C` une nouvelle ligne de commentaires.
— `Prg->Nouveau programme` ou `Alt+P` permettent d'ouvrir non plus une ligne de commandes mais un programme, qu'on exécute ensuite avec `OK` juste au-dessus ou `F9`. Le reste de `Prg` indique les commandes utiles pour les programmes et fonctions.
— Cliquer sur le numéro de ligne puis `Suppr` permet de l'effacer. On peut aussi déplacer les lignes et en sélectionner plusieurs pour effaçage, chercher comment.
— `Ctrl+FlècheHaut` permet de faire répparaître la commande précédente.

2 Prise en main par l'exemple de la ligne de commande

Les exercices suivants sont pour vous familiariser avec des commandes pour des calculs de base (ou comment s'en rappeler en cas de besoin). Après chaque exercice on doit pouvoir répondre aux questions associées (ou bien chercher dans l'aide si nécessaire).

Pour faciliter les entrées :

`;` sépare entre deux instructions sur une même ligne.

`:;` fait de même en n'affichant pas le résultat de l'instruction qui précède.

`Maj+Entrée` permet de passer à la ligne (en mode ligne de commandes) sans exécuter.

`Alt+N` crée une nouvelle ligne de commandes.

`Alt+C` crée une ligne de commentaires si vous voulez prendre des notes pour vous-mêmes ou expliquer la feuille de calculs (ce ne sera pas compilé).

`?` ou `F1` permettent d'accéder à l'index des commandes.

1. Définir le polynôme $P = (x+4)^3(x-5)^2$ et le développer.
   Comment affecter une variable ?
2. Simplifier les expressions suivantes :
   $$\sqrt{3+2\sqrt{2}}, \quad \frac{\sin(3x)+\sin(7x)}{\sin(5x)}, \quad e^{i\pi/6}, \quad 4\arctan(1/5) - \arctan(1/239)$$
   Comment créer une racine cubique, quartique, etc ?
3. Factoriser les polynômes
   $$x^6 - 2x^3 + 1, \quad (-y+x)z^2 - xy^2 + x^2y$$
   Avec la configuration par défaut, comment factoriser dans $\mathbb{C}[X]$?
4. Calculer les primitives (et les exprimer de manière simple) des fonctions qui à x associent
   $$\frac{1}{e^x - 1}, \quad \frac{1}{x\ln(x)}, \quad e^{x^2}, \quad x\sin(x)e^x.$$
   Dériver les expressions obtenues.
5. Calculer les intégrales
   $$\int_1^2 \frac{1}{(1+x^2)^3}dx, \quad \int_1^2 \frac{1}{1+x^3}dx,$$
6. Calculer les sommes
   $$\sum_{k=1}^{N} k, \quad \sum_{k=1}^{N} \frac{1}{k^2}$$
   puis la limite de la deuxième.
7. Calculer le développement de Taylor en $x = 0$ à l'ordre 4 de
   $$\ln(1+x+x^2), \quad \sqrt{1+e^x}, \quad \frac{\ln(1+x)}{e^x - \sin(x)}.$$
   Quel argument faut-il donner pour avoir un terme final $O(x^n)$?
8. Trouver les entiers n tels que le reste de la division euclidienne de $123n$ par 2020 soit 11.
   Quelles sont les commandes de quotient et reste pour les entiers ?
9. Déterminer la liste des diviseurs de 45768 et factoriser 100 !
10. Résoudre le système linéaire avec un paramètre
    $$\begin{cases} x + y + az &= 1 \\ x + ay + z &= 2 \\ ax + y + z &= 3 \end{cases}$$
    Sous quelle forme doit-on présenter les équations ? Et que se passe-t'il pour un paramètre donnant un rang plus petit ?
11. Définir et trouver l'inverse de la matrice
    $$A = \begin{pmatrix} 1 & 1 & 1 & a \\ 1 & 1 & a & 1 \\ 1 & a & 1 & 1 \\ a & 1 & 1 & 1 \end{pmatrix}$$

3 Les objets du calcul formel dans Xcas

Pour toute cette partie, il est fortement recommandé de lire les explications et ensuite d'exécuter soigneusement les instructions en `Xcas` qui sont données, pour comprendre ce qui se passe à chaque fois.

3.1 Les nombres

Les nombres peuvent être exacts ou approchés :

— Les nombres sont exacts lorsque ce sont des expressions avec des entiers et des constantes prédéfinies.

```
sqrt(2)*e^(i*pi/3)
500!
```

— Ils sont approchés (flottants) et alors traités en notation scientifique standard. Dès qu'un nombre approché apparaît le calcul se fait en mode approché, sinon il est en mode exact.

```
1/(105066.1)
pi + 1.5
```

— Pour évaluer un nombre exact, utiliser `evalf`.

```
Digits:=50
evalf(pi)
```

Attention La lettre `i` est réservée au nombre imaginaire, ne pas l'utiliser comme indice de boucle!!

— Xcas sait aussi gérer les infinis (signés ou non) lorsque c'est possible, tester

```
1/0 ; - (1/0)^2 ; 1/infinity ; infinity + 4
infinity/infinity
```

3.2 Les variables

Voici les informations de base concernant les variables :

— Elles sont identifiées par un nom de variable avec un ou plusieurs caractères, de sorte que `ab` est interprété comme la variable `ab` et non pas un éventuel produit de `a` et `b`.

— Xcas fait attention à la casse (majuscules ou non), donc `a` et `A` ne sont pas les mêmes.

— L'affectation se fait par `:=` et on pas `=`, qui est utilisé pour les équations, ni avec `==` qui est l'égalité booléenne.

```
a = 2 ; a + 3
a := 2 ; a + 3
```

— Pour vider une variable, utiliser `purge`. Pour vider toutes les variables affectées, utiliser `restart`

```
purge(a) ; a == 2
a := 3; b := X^2 + 3 ; c := "s" ; restart
```

— On peut faire une hypothèse sur une variable avec `assume`, qui sera alors utilisée dans la mesure du possible.

```
assume(a<0) ; sqrt(a^2)
```

— La commande `subst` permet de remplacer une variable dans une expression (pour l'évaluer) sans modifier la variable

```
a := x^3 + 1
subst(a^2+1, a= 1) ; a
subst(a^2 + 1, a = sqrt(b-1)) ; a
```

Pour les plus à l'aise on peut stocker une valeur dans une variable par référence mémoire (par exemple pour changer la valeur dans une liste) en utilisant `=<` ce qui est plus rapide, mais attention car son usage est piégeux.

3.3 Les expressions

Une expression est une combinaison de nombres et de variables reliés entre eux par des opérations. Si les variables ont une valeur, Xcas les utilise pour évaluer l'expression

```
s := abc+variable+s1f4 ; expand(s^2)
abc := 35 ; s ; variable := 1 ; s
```

L'évaluation simplifie automatiquement via certaines opérations (mise sous forme irréductible, formes trigonométriques basiques, ...). Pour simplifier de manière plus complexe, plusieurs choix :

— `normal` et `ratnormal` (efficaces) écrivent une fraction rationnelle sous forme de fraction irréductible développée, la seconde avec seulement des rationnels.

```
b:=sqrt(1-a^2)/sqrt(1-a) ; ratnormal(b) ; normal(b)
```

— `factor` un peu plus lente, factorise la forme irréductible.

```
c := (x^3 + x^2 + x + 1)/(x^5 -1) ; factor(c)
```

— `simplify` essaie de se ramener à des variables algébriquement indépendantes puis applique `normal`. Est assez lourd et peut demander plusieurs itérations, et/ou des hypothèses `assume`.

```
simplify(b) ; simplify(simplify(b)) ;
```

— `tsimplify` essaie de se ramener à des variables algébriquement indépendantes et c'est tout.

```
tsimplify(b)
```

Concernant les expressions, quelques autres commandes sont particulièrement intéressantes :

— `expand` développe une expression par puissances entières.

```
s := (a*x^2 + b)^3 + c*b^5 ; expand(s)
```

— `convert` permet de passer d'une expression à une autre équivalente en spécifiant le format de sortie :

```
convert(exp(i*x), sincos)  ; convert(cos(x) + 2*sin(x), exp)
convert(1/(x^4-1), partfrac)
convert(series(sin(x),x=0,6), polynom)
```

3.4 Fonctions basiques

— Les fonctions classiques (sur les réels, lse complexes, trigonométriques, hyperboliques) sont toutes définies dans Xcas, consulter l'aide et plus précisément le menu `Cmds`.

— Ensuite, pour créer une fonction, l'utilisateur doit la déclarer grâce à une expression contenant la variable. On a trois manières de procéder, en particulier la commande `unapply` à retenir :

```
f(x) := x^2 - 1 ; f(2) ; f(a^2)
f:=(x,y)->x^2*y-1 ; f(2,3)
s := x^2*y+x +y - 1 ; f := unapply(s,x,y) ; f
```

— On remarque avec la première ligne qu'on peut appliquer une expression `E` à une fonction mais le résultat est alors une expression.
Une différence essentielle entre fonctions et expressions est qu'on ne peut pas évaluer directement une expression :

```
E := x^2 - 1 ; E(2)
E := x^2*y - 2 ; subst(E, x=2, y=3)
```

— On peut dériver une fonction par rapport à une de ses variables avec `diff`, mais attention quand on veut la stocker.

```
f(x):=x^2 - 1
f1(x):=diff(f(x)) à ne surtout pas faire
f1:=f' ou f1:=unapply(diff(f(x),x)) est la bonne méthode.
```

Pour une seule variable, on peut se contenter de `function_diff`.

3.5 Structures de données

3.5.1 Chaînes et caractères

Une chaîne est une suite de caractères entre guillemets " et un caractère est une chaîne à un élément. Attention, l'indexation commence à 0. Les opérations de base sur les chaînes ne les modifient pas implicitement.

```
s:="azertyuiop" ; size(s) ; s[0] ; s[size(s)-1]
concat("qwert",s) ; s
head(s) ; tail(s) ; mid(s,1,4)
```

On peut passer d'un nombre à une chaîne par `string` et d'une chaîne à un nombre (ou une expression) par `expr`

```
s := string(12356); s ; type(s)
expr("1435660") ; expr("3*x+1")
```

3.5.2 Listes, séquences, ensembles

`Xcas` a trois collections d'objets, dont les séparateurs sont toujours les virgules :

- Les listes (entre crochets).
- Les séquences (entre parenthèses).
- Les ensembles (définis via `set[...]`).

```
liste:=[1,2,4,2] ; liste[2] ; liste(1)
sequence:=(1,2,4,2) ; sequence(0) ; sequence[2]
ensemble:=set([1,2,4,2])
```

Attention L'indiciage par `[]` commence à 0 et celui par `()` à 1, il faut simplement éviter de mélanger les deux. Préférer `[]` en cas d'indiciage de symboles pour ne pas confondre avec des fonctions algébriques, ou bien `[[...]]` si on veut absolument commencer à 1.

On peut faire des listes de listes (cf. les matrices), mais les séquences sont plates (ne peuvent avoir comme éléments des séquences). Un ensemble n'a pas d'ordre et chaque objet y est unique.

On peut convertir une séquence en liste en entourant par `[]` (ou via `nop`) et en ensemble en entourant par `set[]`. Pour aller de liste à séquence on utilise `op`. Les longueurs des séquences, listes et tailles d'ensemble ont toutes leurs fonctions associées qui sont `size` ou `nops`.

```
op(liste) ; size(liste)
nop(sequence) ; nops(sequence)
size(ensemble)
```

Pour créer des listes ou séquences, on utilise les commandes d'itération `$` ou `seq` ou encore `makelist`.

```
1$5 ; [1$5]
se := k^2 $ (k=-2..2)
seq(k^2+ 7,k=-3..3)
[k^2$(k=-2..2)]
makelist(x->x^2,-2,2)
makelist(x->x^2,-5,5,2)
```

On peut aussi ajouter des éléments, y compris en famille avec les commandes d'itération précédentes.

```
sequence:=sequence, 37 ; sequence := sequence, k^2$ (k=-2..2)
liste := append((liste,(2*k^2+1)$ (k=-2..2))
```

On peut modifier les listes « en place » et de même pour les séquences.

```
se := (1,2,3) :; se(1) := 12 :; se
li := [1,2,3] :; li[0] := 13 :; li
```

4 Programmation

Le texte pour un programme (ou une fonction un tant soit peu compliquée) ne tient en général pas sur une ligne, on utilise donc un éditeur de programmes, via `Prg->Nouveau programme` ou `Alt+P`.

Pour les structures de programmation habituelles (qu'on détaille ci-dessous), consulter en cas de besoin les menus `Fonctions, Test, Boucles, Prg->Ajouter`.

Il est possible de saisir des programmes avec plusieurs syntaxe différentes, réglables dans la `Configuration du CAS` :

- `Xcas` accepte un style algorithmique en français (celui que j'emploierai toujours plus bas) ainsi qu'une syntaxe type C.
- `Python` (avec un choix pour les puissances) accepte un style proche du python (on donnera les instructions en python « pur », à tester sous Xcas lui-même)
- `Maple, Mupad, TI89/92` ont le sens qu'on imagine, mais ce ne sera pas exploré cette année.
  À noter qu'un programme défini par disons `f` peut être converti des syntaxes `Xcas` vers `Python` via `python(f)` et réciproquement par `xcas(f)`.
  Important Xcas est souple mais il faut éviter de mélanger plusieurs syntaxes, donc s'en fixer une au début !

Dans tous les cas, la touche `Entree` ne fait que passer à la ligne dans ce mode, et il faut donc utiliser `OK` ou `F9` pour exécuter.

4.1 Tests

4.1.1 Égalité, comparaison et opérateurs booléens

Pour tester l'égalité ou la différence de 2 expressions, on utilise respectivement == et <> ou != . Attention, le test ne simplifie pas forcément les expressions, donc utiliser `simplify` pour assurer le coup. Pour l'ordre on utilise <=,<,>=.

```
2 == 0 ; 2 != 0 ; 2 <> 2
a := 2 ; a ==2 ; ((a+1)^2 - 2*a-1) == 0
a>2 ; a<2 ; a<>2
```

En fait, le résultat d'un test (même si affiché comme booléen) est 1 si le test est vrai, et 0 s'il est faux. Réciproquement, on peut utiliser 1 et 0 comme booléens.

```
(2==2) + 1
(2<3) - 1
```

Les opérateurs booléens sont `and, et, &&` (et logique) et `or, ou, ||` (ou logique), la négation logique est `not(), non(), !`.

```
2==3 ou 2 ==2 ; 2==2 et 3==2
```

4.1.2 Tests "si... alors"

Pour exécuter une instruction après test booléen, la syntaxe (en Xcas) est l'une des suivantes :

```
si ... alors ... (;) sinon ... (;) fsi;
if ... then ... else ... fi;
x := -2 ; si x >0 alors y:=x ; sinon y:=-x ; fsi ; y
```

Dans le premier cas, les séparateurs ; ne sont pas obligatoires mais conseillés pour bien marquer la fin des instructions. Il est également conseillé d'indenter à l'usage dans un programme (même si ce n'est pas crucial pour Xcas).

4.2 Boucles

On peut exécuter des instructions à répétition en utilisant une boucle définie `pour` (ou `for` en syntaxe type C) ou une boucle indéfinie `tantque` (ou `while` en syntaxe type C).

```
pour ... de ... jusque ... faire ... fpour ;
f := 1 ; pour j de 1 jusque 10 faire f:=f*j; fpour;
```

Attention Ne pas utiliser `i` comme indice de boucle, il est réservé comme constante imaginaire pure.

```
tantque ... faire ... ftantque;
a := 356 ; b := 157
tantque b!=0 faire r:=irem(a,b) ; a:=b :; b:=r :; ftantque; a
```

4.3 Fonctions

On doit souvent définir une fonction après une suite d'instructions. La syntaxe en Xcas est la suivante :

```
fonction nom_fonction(variables)   (ou nom_fonction(variables):={)
    local (variables_locales);
    (suite_d_instructions) ;
    retourne (resultat);
ffonction;                          (avec alors }:;))
```

Par exemple, voici une fonction de calcul de pgcd.

```
pgcd(a,b):={
     local r;
     tantque b!=0 faire
         r:=irem(a,b);
         a:=b;
         b:=r;
     ftantque;
     retourne a;
}:;
```

À noter que les variables arguments de la fonction sont muettes : si on avait déjà défini `a` et `b` hors du programme plus haut, elles ne sont pas modifiées lorsqu'on exécute `pgcd` (et de même, `r` n'est pas attribuée si elle ne l'était pas avant d'exécuter le programme).

Pour comprendre ou débugger un programme en l'exécutant pas à pas, on utilise la commande `debug` en l'appliquant à une exécution du programme.

```
debug(pgcd(7,13))
```

5 Exercices pratiques

Sauf indication précise, on ne vise (pour l'instant) pas des algorithmes aux performances optimales, le but étant de se familiariser avec la programmation sous Xcas.

1. Algorithmes fondamentaux : écrire des programmes implémentant
   (a) l'écriture d'un entier n en base b.
   (b) le pgcd de 2 entiers a et b en utilisant la valeur absolue du reste symétrique.
   (c) l'algorithme de Bézout (qui pour a et b renvoie u, v tels que $au+bv = 1$).
   (d) l'inverse modulaire d'un entier a modulo n en ne calculant que ce qui est nécessaire dans Bézout.
   (e) les restes chinois : pour m et n premiers entre eux et a modulo m, b modulo n, donner l'entier k congru à a modulo m et b modulo n.
   (f) le test de primalité naïf d'un entier n par division.
   Pour chacun d'entre eux, trouver la fonction intégrée correspondante et comparer les résultats (cf. `convert, gcd, iegcd,inv, ichinrem`).
2. Comparer grâce à la commande `time` le temps de calcul de a^n mod m par la fonction `powmod` et celui du calcul du reste modulo m après avoir calculé a^n et pris le reste modulo m. Programmez ces deux méthodes (en exponentiation rapide) et comparez de nouveau les temps.
3. Déterminer le plus grand $x = 2^(-n)$ avec $n \in \mathbb{N}$ tel que $(1.0 + x) - 1.0$ renvoie 0 avec la précision par défaut puis avec `Digits:=30`.
4. Calculer la valeur de $\exp(\pi\sqrt{163})$ avec 30 chiffres significatifs. Proposer une commande permettant de décider si un réel est un entier.
5. Déterminer la valeur et le signe de la fraction rationnelle
$$F(x,y) = \frac{1335}{4}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11}{2}y^8 + \frac{x}{2y}$$
   où $x = 77617$ et $y = 33096$, par calcul approché puis par calcul exact. Comparer les résultats. Combien de chiffres significatifs faut-il pour arriver à une bonne valeur en mode approché ?
6. Que se passe-t'il si on veut utiliser l'exponentiation rapide pour calculer $(x+y+z+1)^k$ pour $k = 100$ par exemple ? Calculer le nombre de termes du développement et expliquer.
7. Utiliser `type` ou `whattype` pour déterminer comment le logiciel représente une fraction, un nombre complexe, un flottant en précision machine, la variable x, l'expression $\sin(x) + 2$, la fonction `x->sin(x)`, une liste, une séquence, etc...