1 TP de prise en main

Nous utiliserons Xcas, que l'on lance sur les netbooks depuis le menu Education (pensez à mettre Xcas à jour sur les netbooks empruntés, connectez-vous à Internet puis lancez Xcas et cliquez sur le menu Fich, Quitter et mettre à jour), sur les PC du DLST a partir de l'icone Xcas du bureau. Vous pouvez aussi l'installer si vous disposez de votre propre PC ou Mac, cf. `www-fourier.ujf-grenoble.fr/~parisse/install_fr`.

Si vous n'avez jamais utilisé Xcas auparavant, commencez par parcourir le tutoriel, celui-ci est accessible depuis le menu `Aide, Debuter en calcul formel`. Lisez au moins jusqu'au paragraphe 2.4 (vous pouvez sauter le paragraphe 2.2 sur les chaines de caractère), puis faites les exercices du TP de prise en main. Pour trouver les noms de commandes, regardez d'abord dans le menu Outils, puis Cmds si vous ne trouvez pas dans le menu Outils, ou dans le menu Graphe qui fournit des assistants pour les représentations graphiques.

Les TP qui suivent ce TP de prise en main demandent souvent d'écrire des petits programmes, pour cela vous pouvez consulter le paragraphe 6 du tutoriel.

1. Écrire le polynôme $(x+3)^7 \times (x-5)^6$ selon les puissances décroissantes de x.
2. Simplifier les expressions suivantes :
$$\sqrt{3+2\sqrt{2}}, \quad \frac{1+\sqrt{2}}{1+2\sqrt{2}}, \quad e^{i\pi/6}, \quad 4\text{atan}(\frac{1}{5}) - \text{atan}(\frac{1}{239})$$
3. Factoriser :
$$x^8 - 3x^7 - 25x^6 + 99x^5 + 60x^4 - 756x^3 + 1328x^2 - 960x + 256$$
$$x^6 - 2x^3 + 1, \quad (-y+x)z^2 - xy^2 + x^2y$$
4. Calculez les intégrales et simplifiez le résultat :
$$\int \frac{1}{e^x - 1}\, dx, \quad \int \frac{1}{x\ln(x)} \ln(\ln(x))dx, \quad \int e^{x^2}\, dx, \quad \int x\sin(x)e^x\, dx$$
Vérifiez en dérivant les expressions obtenues.
5. Déterminer la valeur de :
$$\int_1^2 \frac{1}{(1+x^2)^3}, \quad \int_1^2 \frac{1}{x^3+1}\, dx$$
6. Calculer les sommes suivantes
$$\sum_{k=1}^{N} k, \ \sum_{k=1}^{N} k^2, \ \sum_{k=1}^{\infty} \frac{1}{k^2}$$
7. Développer $\sin(3x)$, linéariser l'expression obtenue et vérifier qu'on retrouve l'expression initiale.
8. Calculer le développement de Taylor en $x=0$ à l'ordre 4 de :
$$\ln(1+x+x^2), \quad \frac{\exp(\sin(x))-1}{x+x^2}, \quad \sqrt{1+e^x}, \quad \frac{\ln(1+x)}{\exp(x)-\sin(x)}$$
9. Trouver les entiers n tels que le reste de la division euclidienne de $123n$ par 256 soit 17.

10. Déterminer la liste des diviseurs de 45768.
    Factoriser 100 !
11. Résoudre le système linéaire :
$$\left\{\begin{array}{rcrcrcl} x & + & y & + & az & = & 1 \\ x & + & ay & + & z & = & 2 \\ ax & + & y & + & z & = & 3 \end{array}\right.$$
12. Déterminer l'inverse de la matrice :
$$A = \begin{pmatrix} 1 & 1 & 1 & a \\ 1 & 1 & a & 1 \\ 1 & a & 1 & 1 \\ a & 1 & 1 & 1 \end{pmatrix}$$

2 TP1 : Suites récurrentes

On souhaite étudier des suites récurrentes (u_n) définies par $u_{n+1} = f(u_n)$ et u_0, où f est une fonction de $\mathbb{R}$ dans $\mathbb{R}$, par exemple $f(x) = \sqrt{2+x}$.

Exercice 1

1. En utilisant le tableur de Xcas (menu Tableur), représenter les premiers termes de la suite $u_{n+1} = \sqrt{2+u_n}$ avec $u_0 = 0.1$ ou $u_0 = 1/10$ (menu Math->suite récurrente du tableur).
2. Calculer ensuite en mode exact et approché les 10 premiers termes de la suite : vous pouvez définir la cellule `A1` par `=sqrt(2+A0)`, puis déplacer la souris vers la partie situé en bas à droite de la cellule `A1` (le curseur souris change de forme), puis appuyer sur le bouton de la souris et relâcher à la fin de la zone où vous voulez copier la formule définissant `A1`.
3. Quels sont les avantages et inconvénients d'utiliser une valeur initiale exacte ou approchée ?

La feuille de calcul précédente donne une idée de la convergence de la suite, mais ne donne aucune information quantitative sur la vitesse de convergence. Au lieu de calculer un nombre fixé de termes de la suite, on va écrire un programme avec un test d'arrêt selon la valeur $|u_{n+1} - u_n|$ comparé à un nombre positif (petit) ε fixé à l'avance. Pour éviter que le programme ne boucle indéfiniment lorsque la suite ne converge pas (ou converge trop lentement pour la machine), on fixe aussi un nombre maximal d'itération N.

Exercice 2

Écrire un programme `iter` prenant en argument la fonction f, la valeur de u_0, de N et de ε, et qui s'arrête dès que l'une des conditions suivantes est satisfaite :

- $|u_{n+1} - u_n| < \varepsilon$
- le nombre d'itérations dépasse N.

Dans le premier cas le programme renverra la valeur de u_{n+1}, dans le second cas une séquence composée de u_N et de N.
Tester votre programme avec $f(x) = \sqrt{2+x}$ et $f(x) = x^2$.

On suppose que la fonction f satisfait aux hypothèses du théorème du point fixe. On notera $k < 1$ la constante de contractance. On peut alors trouver un encadrement de la limite l de la suite (u_n) en fonction de u_n, u_{n-1} et k.

Exercice 3

Écrire un programme `iter_k` prenant en argument la fonction f, la valeur de u_0, la constante k et l'écart toléré ε, et qui s'arrête dès que $|u_n - l| \leq \varepsilon$.

Vérifier les hypothèses du théorème du point fixe pour $f(x) = 2\cos(x/3)$ sur $[0, 2]$ et expliciter une constante de contractance k. Déterminer une valeur approchée de la limite de (u_n) à `1e-3` près en utilisant la fonction `iter_k`.

La convergence de ces suites est en général linéaire*, le nombre de décimales exactes augmente de la même valeur à chaque itération. Par contre lorsqu'on est prêt d'une racine, la méthode de Newton permet en gros de multiplier par deux le nombre de décimales à chaque itération.*

Exercice 4

1. Donner une suite itérative obtenue par la méthode de Newton convergeant vers $\sqrt{7}$.
2. Montrer que la fonction $f : \mathbb{R}_+ \to \mathbb{R}_+$ définie par

$$f(x) = \frac{7x+7}{x+7}$$

   admet $\sqrt{7}$ pour point fixe. Trouver un intervalle I contenant $\sqrt{7}$ sur lequel les hypothèses du théorème du point fixe sont satisfaites et expliciter une constante de contractance k.
3. Comparer au tableur la vitesse de convergence des 10 premiers termes des deux suites (calculez les avec par exemple 200 chiffres significatifs et faites `evalf(.,20)` sur les différence entre 2 termes successifs).
4. En utilisant la fonction `iter`, trouver un encadrement de $\sqrt{7}$ à `1e-6` près par les deux méthodes (on pourra prendre une valeur initiale approchée puis entière exacte pour avoir une valeur numérique approchée puis une fraction). Combien d'itérations sont nécessaires ?

Dans certains cas, la fonction f n'est pas contractante, mais on peut réécrire l'équation à résoudre sous une autre forme avec une fonction contractante, par exemple en utilisant une fonction réciproque.

Exercice 5

Donner un encadrement à `1e-6` près d'une racine de l'équation $\tan(x) = x$ sur l'intervalle $]3\pi/2, 5\pi/2[$ en utilisant une méthode de point fixe. On observera qut $\tan$ n'est pas contractante mais que sa fonction réciproque l'est (attention à y ajuster correctement un multiple entier de π).

3 TP2 : Types. Calcul exact et approché. Algorithmes de bases

1. Utiliser la commande `type` pour déterminer la représentation utilisée par le logiciel pour représenter une fraction, un nombre complexe, un flottant en précision machine, un flottant avec 100 décimales, la variable x, l'expression $\sin(x) + 2$, la fonction `x->sin(x)`, une liste, une séquence, un vecteur, une matrice. Essayez d'accéder aux parties de l'objet pour les objets composites (en utilisant `op` par exemple).
2. Déterminer le plus petit entier n tel que $(1.0 + 2^{-n}) - 1.0$ renvoie 0 sur PC avec la précision par défaut puis avec `Digits:=30` (on pourra au préalable mettre `epsilon` à 0 dans la configuration du CAS). Même question sur votre calculatrice si vous en avez une.
3. Calculer la valeur de $a := \exp(\pi\sqrt{163})$ avec 30 chiffres significatifs, puis sa partie fractionnaire (`frac(.)`). Proposez une commande permettant de décider si a est un entier.
4. Déterminer la valeur et le signe de la fraction rationnelle

$$F(x,y) = \frac{1335}{4}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11}{2}y^8 + \frac{x}{2y}$$

   en $x = 77617$ et $y = 33096$ en faisant deux calculs, l'un en mode approché (`F(77617.0,33096.0)`) et l'autre en mode exact. Que pensez-vous de ces résultats ? Combien de chiffres significatifs faut-il pour obtenir un résultat raisonnable en mode approché ?
5. À quelle vitesse votre logiciel multiplie-t-il des grands entiers (en fonction du nombre de chiffres) ? On pourra tester le temps de calcul $t(n)$ (`time(a*(a+1))[0]`) du produit de $a \times (a+1)$ pour $a = 10^n$ avec $n = 10000, 20000, 40000$. Comment évolue le temps de calcul lorsque le nombre de décimales double ?
6. Comparer le temps de calcul de $a^n \pmod m$ par la fonction `powmod` et la méthode "prendre le reste modulo m après avoir calculé a^n" (vous pouvez aussi programmer la méthode rapide et la méthode lente, cf. par exemple l'article exponentation rapide de wikipedia).

7. Programmation de la méthode de Horner
   Il s'agit d'évaluer efficacement un polynôme

$$P(X) = a_n X^n + ... + a_0$$

   en un point. On pose $b_0 = P(\alpha)$ et on écrit :

$$P(X) - b_0 = (X - \alpha)Q(X)$$

   où :

$$Q(X) = b_n X^{n-1} + ... + b_2 X + b_1$$

   On calcule alors par ordre décroissant b_n, b_{n-1}, ..., b_0.

   (a) Donner b_n en fonction de a_n puis pour $i \leq n-1$, b_i en fonction de a_i et b_{i+1}. Indiquez le détail des calculs pour $P(X) = X^3 - 2X + 5$ et une valeur de α entière non nulle.

   (b) Écrire un fonction `horn` effectuant ce calcul : on donnera en arguments le polynôme sous forme de la liste de ces coefficients (dans l'exemple `[1,0,-2,5]`) et la valeur de α et le programme renverra $P(\alpha)$. (On pourra aussi renvoyer les coefficients de Q).

   (c) En utilisant cette fonction, écrire une fonction qui calcule le développement de Taylor complet d'un polynôme en un point.

4 TP3 : Séries entières

Exercice 1.

1. Rappeler le développement de Taylor $T_{2n+1}(x)$ au voisinage de $x = 0$ de $f(x) = \sin(x)$ à l'ordre $2n$.
2. Tracer sur un même graphique les graphes des fonctions f et T_1, T_3, T_5, T_7
3. Graphiquement on voit que $T_7(x)$ approche $\sin(x)$: sur quel intervalle cette approximation vous paraît-elle acceptable ?
4. Donner une majoration du reste $R_{2n+1}(x)$ du développement de Taylor de f à l'ordre $2n+1$, où $f(x) = T_{2n+1}(x) + R_{2n+1}(x)$.
5. On prend $T_7(x)$ comme valeur approchée de $\sin(x)$ pour $x \in [-1, 1]$.
   Donner une majoration indépendante de x de l'erreur commise.
   (A titre d'illustration, tracer la différence $T_7(x) - \sin(x)$.)
6. En déduire un encadrement de $\sin(1)$.

Exercice 2. On veut approcher $\cos(x)$ à `1e-6` près en utilisant des développements en séries entières.

1. Déterminer le plus petit k tel que :

$$T_{2k}(x) = \sum_{j=0}^{k} (-1)^j \frac{x^{2j}}{(2j)!}$$

   réalise cette approximation sur $[0, \pi/4]$.
2. Écrire une fonction qui calcule une valeur approchée à `1e-6` de $\cos(x)$ sur $[-100, 100]$ en justifiant et en effectuant les étapes suivantes :
   – on retire un multiple entier de π (obtenu par arrondi de x/π) pour se ramener à l'intervalle $[-\pi/2, \pi/2]$ (on discutera sur l'erreur commise)
   – si x est négatif, on remplace x par $-x$ (que devient $\cos(x)$?)
   – lorsque $x \in [0, \pi/4]$, on utilise le développement en séries ci-dessus.
   – lorsque $x \in [\pi/4, \pi/2]$, on se ramène au développement de l'exercice précédent en appliquant la formule $\sin(x) = \cos(\pi/2 - x)$.
3. Afin de tester votre fonction f et éviter d'éventuelles erreurs grossières, faites afficher le graphe de f, disons sur l'intervalle $[-10, 10]$, puis le graphe de la différence $f - \cos$ où $\cos$ est la fonction déjà implémentée dans Xcas.

Exercice 3

1. Pour $x > 0$ exprimer $\arctan(-x)$ en fonction de $\arctan(x)$. Calculer la dérivée de $\arctan(x)+\arctan(1/x)$, en déduire $\arctan(1/x)$ en fonction de $\arctan(x)$ pour $x > 0$. Montrer que le calcul de $\arctan(x)$ sur $\mathbb{R}$ peut se ramener au calcul de $\arctan(x)$ sur $[0, 1]$.
2. Rappeler le développement en séries entières de $\arctan(x)$ en $x = 0$, et son rayon de convergence. Soit $\alpha \in [0, 1]$, montrer que

$$\alpha - \frac{\alpha^3}{3} + \frac{\alpha^5}{5} - \frac{\alpha^7}{7} \leq \arctan(\alpha) \leq \alpha - \frac{\alpha^3}{3} + \frac{\alpha^5}{5}$$

   en déduire que la méthode de Newton appliquée à l'équation $\tan(x) - \alpha = 0$ avec comme valeur initiale $\alpha - \frac{\alpha^3}{3} + \frac{\alpha^5}{5}$ est une suite décroissante qui converge vers $\arctan(\alpha)$.
3. Déterminez par cette méthode une valeur approchée à `1e-8` près de $\pi/4 = \arctan(1)$.
4. On pourrait calculer $\pi/4$ avec la même précision en utilisant le développement en séries de la formule de Machin

$$\frac{\pi}{4} = 4\arctan(\frac{1}{5}) - \arctan(\frac{1}{239})$$

   Combien de termes faudrait-il calculer dans le développement des deux arctangentes ?

Exercice 4

1. Écrire le développement en séries entières au voisinage de $x = 0$ de :

$$g(x) = \frac{e^{-x} - 1}{x}$$

2. On veut calculer une valeur approchée de

$$I = \int_0^1 g(x)\, dx$$

   En utilisant le développement de g, écrire I sous la forme d'une série $\sum_{j=0}^{\infty} v_j$.
3. Soit $R_n = \sum_{j=n+1}^{\infty} v_j$ le reste de cette série. Donner une majoration de $|R_n|$.
4. En déduire un encadrement de I faisant intervenir $\sum_{j=0}^{n} v_j$. Calculer explicitement cet encadrement lorsque $n = 10$.

Exercice 5

Cet exercice reprend les calculs de $\ln(x)$ et $\exp(x)$ discutés en cours dans l'objectif de les illustrer par vos propres expériences sur ordinateur.

1. La série alternée $s_n = \sum_{k=1}^{n} (-1)^{k+1} \frac{1}{k}$ tend vers $\ln 2$, et les termes consécutifs donnent des encadrements $s_{2m+1} < \ln 2 < s_{2m}$. Jusqu'à quel rang faut-il aller afin de garantir un encadrement à 10^{-5} près ? Calculer cette approximation de $\ln 2$ avec Xcas. Même question pour 10^{-10}. Qu'observez-vous ?

   On considère ensuite la série $\ln 2 = \sum_{k=0}^{\infty} \frac{2}{(2k+1)3^{2k+1}}$. Jusqu'à quel rang faut-il aller afin de garantir une approximation à 10^{-5} près ? Calculer cette approximation de $\ln 2$ avec Xcas. Même question pour 10^{-10}. Conclusion ?
2. On se propose d'approcher $\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ pour $x = -32$. Avant de se lancer dans ce calcul, estimer l'ordre de grandeur de $\exp(-32)$ puis l'ordre de grandeur des plus grands termes dans la somme.

   Jusqu'à quel rang faut-il aller afin de garantir une approximation à 10^{-20} près ? Calculer cette approximation de $\exp(-32)$ avec Xcas, d'abord en utilisant la précision standard de 53 bits, soit 16 décimales. Quel problème observez-vous ? On pourra augmenter la précision des nombres flottants utilisés : quelle précision est nécessaire, environ, pour raisonnablement effectuer ce calcul ?

   Est-ce que ces problèmes se posent pour l'approximation de $a_0 = \exp(-1)$? Comment en déduire une approximation de $\exp(-32)$ avec un minimum d'opérations ?

Exercice 6

On reprend des idées des exercices 4 et 5 pour implémenter le calcul de la fonction sinus intégral définie par :

$$\mathbf{Si}(x) = \int_0^x \frac{\sin(t)}{t}\, dt$$

1. Donner un développement en séries entières de la fonction **Si**, ainsi que le rayon de convergence, et une majoration du reste en fonction de x.
2. Lorsque $x > 1$, on a le problème de la série qui "commence par diverger" (comme pour l'exponentielle de l'exercice 5.2), mais sans possibilité de "réduire l'argument". Combien de chiffres significatifs perd-on lorsqu'on calcule **Si**(x) par la série entière ? (En pratique, on calculera la série avec ce nombre de chiffres significatifs en plus).
3. Lorsque x est grand, on préfère utiliser le développement asymptotique de la fonction **Si**. On admettra que

$$\int_0^{+\infty} \frac{\sin(t)}{t}\, dt = \frac{\pi}{2}$$

   On calcule alors **Si** en faisant des intégrations par parties successives, en intégrant la fonction trigonométrique et en dérivant la fraction rationnelle. Les termes tout intégré donnent le développement asymptotique, et l'intégrale le reste. Par exemple, calculer le développement asymptotique à l'ordre 10 et une majoration du reste pour $|x| \geq 100$. En déduire une valeur approchée de **Si**(100).
4. En calculant **Si**(100) à l'aide du développement en séries entières, en déduire une valeur approchée de π.

5 TP4 : Polynômes

Exercice 1

Donner le détail des calculs avec Bézout de la décomposition en éléments simples de :

$$f(x) = \frac{1}{(x^2-4)(x-1)}$$

en déduire :

- une primitive de f
- le coefficient de x^n du développement en séries entières de cette fraction en 0.

Exercice 2

Factoriser sur $\mathbb{R}$ (`factor`) et sur $\mathbb{C}$ (`cfactor`) le polynôme $P(x) = x^6 + x^4 + x^3 + x^2 + x + 1$. Quels sont les degrés des facteurs ? Même question mais en remplacant P par `evalf(P)`. En regroupant les racines de P, retrouver la factorisation exacte de P dans $\mathbb{R}$ et $\mathbb{C}$.

Exercice 3

Trouver une racine du polynôme P ci-dessus en appliquant la méthode de Newton avec une valeur initiale complexe aléatoire, éliminer cette racine par division euclidienne, chercher une autre racine, etc. jusqu'à obtenir la factorisation complète de P. Comparer la valeur de P et celle obtenue en développant le produit des $X-$racine.
Bonus : Écrire une fonction qui cherche une racine d'un polynôme en utilisant la méthode de Newton. Ajouter un test pour être sûr que la racine du polynôme est simple. Tester avec le polynôme P ci-dessus. Bonus : modifier la fonction ci-dessus pour trouver toutes les racines (lorsqu'on trouve une racine, on divise le polynôme par $X - \alpha$, et on relance la recherche de racines sur le polynôme obtenu).

Exercice 4

Écrire une fonction qui détermine les racines rationnelles d'un polynôme P à coefficients entiers (elles sont de la forme p/q où q divise le coefficient dominant de P et $\pm p$ divise son coefficient de plus bas degré). Tester avec le polynôme $P = 12x^5 + 10x^4 - 6x^3 + 11x^2 - x - 6$.

Exercice 5

En utilisant les suites de Sturm, déterminer le nombre de racines du polynôme P ci-dessus sur l'intervalle $[-3, 0]$. Même question sur $\mathbb{R}$ tout entier.

Exercice 6

Représenter sur un même graphe $\cos(x)$ et son polynôme interpolateur de Lagrange en utilisant les 7 points d'abscisses équidistantes $\{0, \pi/6, ..., \pi\}$. Donner une majoration de l'erreur entre ce polynome et la fonction $\cos$ en un réel x, représenter graphiquement cette erreur pour $x \in [0, \pi]$. Où l'erreur est-elle la plus grande ?

Exercice supplémentaire

Écrire un programme calculant les coefficients du polynôme d'interpolation de Lagrange par l'algorithme des différences divisées.

Exercice 7

Éliminer successivement a et b du système :

$$\begin{cases} a^3 + b^3 + c^3 &= 8 \\ a^2 + b^2 + c^2 &= 6 \\ a + b + 2c &= 4 \end{cases}$$

puis trouver les racines du polynôme en c, puis calculer les valeurs de b puis a correspondantes (en cherchant les racines du PGCD des 2 puis 3 équations en b puis a après remplacement de c puis b par leurs valeurs).

6 TP5 : Intégration

Exercice 1 : Calculer une valeur approchée de

$$\int_0^1 \frac{dx}{1+x}$$

par la méthode des rectangles, du point milieu et des trapèzes en utilisant un pas de $1/10$ et de $1/100$ (vous pouvez la fonction `plotarea` ou utiliser le tableur ou écrire un programme effectuant ce calcul avec comme arguments la fonction, la borne inférieure, la borne supérieure et le nombre de subdivision). Observez numériquement la différence entre les valeurs obtenues et la valeur exacte de l'intégrale.

Exercice 2

Calculer le polynôme interpolateur P de Lagrange de $f(x) = \frac{1}{1+x^2}$ aux points d'abscisse $\frac{j}{4}$ pour j variant de 0 à 4. Donner un majorant de la différence entre P et f en un point $x \in [0, 1]$. Représenter graphiquement ce majorant. Calculer une majoration de l'erreur entre l'intégrale de f et l'intégrale de P sur $[0, 1]$. En déduire un encadrement de $\pi/4$.

Exercice 3

On reprend le calcul de $\int_0^1 \frac{dx}{1+x}$ mais en utilisant un polynôme interpolateur de degré 4 sur N subdivisions de $[0, 1]$ (de pas $h = 1/N$). Déterminer une valeur de N telle que la valeur approchée de l'intégrale ainsi calculée soit proche à 10^{-8} près de $\ln(2)$. En déduire une valeur approchée à 10^{-8} de $\ln(2)$.

Même question pour $\int_0^1 \frac{dx}{1+x^2}$ et $\pi/4$ (pour majorer la dérivée n-ième de $\frac{1}{1+x^2}$, on pourra utiliser une décomposition en éléments simples sur $\mathbb{C}$).

TP de mathématiques.

Licence 2-Mat 249

2015/16

Table des matières

7 Introduction

Un logiciel de calcul formel permet de faire des manipulations algébriques sans avoir à effectuer d'approximation numérique, par exemple calculer la dérivée ou la primitive d'une fonction, résoudre certaines équations différentielles, etc. Il permet également de faire des calculs numériques, ainsi que des représentations graphiques, ... On en trouve :

- Sur ordinateur :
  Les logiciels les plus utilisés aujourd'hui sont probablement Maple et Mathematica. Ce sont des logiciels commerciaux, vous devez payer une licence d'utilisation si vous les utilisez chez vous. Il existe d'autres logiciels de calcul formel, ceux disponibles à l'agrégation de mathématiques sont Axiom (libre), Gap (libre, spécialisé), Giac/Xcas (libre), Mupad (commercial), Maxima (libre), Pari-GP (libre, spécialisé), Sage (libre). Nous utiliserons Xcas qui est libre (donc gratuit) et de plus compatible avec Maple (pour ceux ayant déjà utilisé Maple, vous pourrez la plupart du temps utiliser les mêmes instructions et le même langage de programmation). Vous pouvez le télécharger à l'URL
  `www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html`
- Sur calculatrices
  Les modèles suivants sont capables de faire du calcul formel
  Casio Graph 100 (incomplet), Casio Classpad, HP 40G(S), HP49G(+), HP50G, TI89 (Titanium), TI 92(+), TI Voyage 200, TI Nspire CAS.

8 Déroulement d'une session.

8.1 Lancement de Xcas

- Sous Windows en installation locale, on clique sur l'icone `xcasfr` du bureau.
- Sous Linux avec Gnome, on clique sur Xcas du menu Education (ou sur l'icone de Xcas si elle apparait). Sinon, ouvrir un terminal et taper `xcas &`.
- sur Mac, cliquez sur Xcas dans le menu Applications du Finder.
- en salle B118 :
  1. Si votre PC est éteint, commencez par appuyer sur les boutons "power" de l'écran et de la tour et patientez une bonne minute.
  2. Connectez-vous sur Windows. Puis lancez Virtualbox.
  3. Si vous n'avez pas de fenêtre de commandes (`Konsole` ou `xterm`), essayez d'en faire apparaitre une :
     - en cliquant sur une icône représentant un écran
     - en cliquant avec le bouton droit, sélectionnez "Exécuter une commande" taper alors `xterm`.
     - en cherchant dans les menus une application Terminal (dans Applications ou Système, Shells ou Accessoires...)

Dans la suite, lorsqu'on demande de taper une commande, il faut saisir la commande puis valider en tapant sur la touche Entree (ou Enter sur un clavier qwerty).

4. Tapez la commande
   `xcas &`

Lors de la première utilisation, choisissez `Xcas` lorsqu'on vous demande de choisir une syntaxe (sauf si vous connaissez le langage Maple). Nous donnons ici seulement le minimum de l'interface à connaitre pour commencer à programmer. On consultera plutot le manuel Débuter en calcul formel ou les autres manuels (menu Aide) pour apprendre à utiliser les fonctionnalités de Xcas en calcul formel, géométrie, tableur, etc..

L'interface apparaît comme suit au lancement de `Xcas`.

Vous pouvez la redimensionner. De haut en bas cette interface fait apparaître :

- une barre de menu gris cliquable : `Fich`, `Edit`, `Cfg`, `Aide`, `CAS`, `Tableur`, `Graphe`, `Geo`,...
- un onglet indiquant le nom de la session, ou `Unnamed` si la session n'a pas encore été sauvegardée,
- une zone de gestion de la session avec un bouton `?` pour ouvrir l'index de l'aide un bouton `Save` pour sauvegarder la session, un bouton `Config: exact real ...` affichant la configuration et permettant de la modifier, un bouton `STOP` permettant d'interrompre un calcul trop long, un bouton `Kbd` pour faire apparaitre un clavier ressemblant à celui d'une calculatrice, qui peut faciliter vos saisies, et un bouton `x` pour fermer la session
- une zone rectangulaire blanche numérotée 1 (première ligne de commande) dans laquelle vous pouvez taper votre première commande (cliquez si nécessaire pour faire apparaitre le curseur dans cette ligne de commande) : `1+1`, suivi de la touche "Entrée" ("Enter" ou "Return" selon les claviers). Le résultat apparaît au-dessous, et une nouvelle ligne de commande s'ouvre, numérotée 2.

Vous pouvez modifier l'aspect de l'interface et sauvegarder vos modifications pour les utilisations futures (menu `Cfg`).

Vous n'avez pour l'instant qu'à entrer des commandes dans les lignes de commandes successives. Si vous utilisez la version html de ce cours, vous pouvez copier-coller les commandes proposées depuis votre navigateur. Chaque ligne de commande saisie est exécutée par la touche "Entrée". Essayez par exemple d'exécuter les commandes suivantes.

```
1/3+1/4
sqrt(2)^5
resoudre(x+3=1,x)
50!
```

Toutes les commandes sont gardées en mémoire. Vous pouvez donc remonter dans l'historique de votre session pour modifier des commandes antérieures. Essayez par exemple de changer les commandes précédentes en :

```
1/3+3/4
sqrt(5)^2
resoudre(2*x+3=0)
500!
```

Notez que

– pour effacer une ligne de commande, on clique dans le numéro de niveau à gauche de la ligne de commande, qui apparait alors en surbrillance. On appuie ensuite sur la touche d'effacement. On peut aussi déplacer une ligne de commande avec la souris.
– Le menu `Edit` vous permet de préparer des sessions plus élaborées qu'une simple succession de commandes. Vous pouvez créer des groupes de lignes de commandes (sections), ajouter des commentaires ou fusionner des niveaux en un seul niveau.
– Le menu `Prg` contient la plupart des instructions utiles pour programmer.

8.2 Aide en ligne

Les commandes de Xcas sont regroupées par thèmes dans les menus du bandeau gris supérieur : `CAS`, `Graphic`, `Geo`, `Cmds`, `Phys`, ... Lorsqu'on sélectionne une commande dans un menu,

- soit l'index de l'aide s'ouvre à la commande sélectionnée (par exemple pour les commandes du menu `CAS`). Cliquez sur le bouton `Details` pour afficher la page du manuel correspondant à la commande dans votre navigateur.
- soit une boite de dialogue s'ouvre vous permettant de spécifier les arguments de la commande (par exemple pour tracer une courbe depuis le menu `Graphic`)
- soit la commande est recopiée dans la ligne de commande. Pour connaitre la syntaxe de cette commande, appuyez sur le bouton `?` en haut à gauche, ou faites afficher la zone de `Messages` (en utilisant le menu `Cfg`,). Vous pouvez aussi configurer Xcas (menu `Cfg` puis `Configuration generale` puis cocher la case `Aide HTML automatique`) pour que la page correspondante du manuel s'ouvre automatiquement dans votre navigateur.

Le menu `Aide` contient les différentes formes d'aide possible : un guide de l'utilisateur (interface), un guide de référence (`Manuels->Calcul formel`, aide detaillée sur chaque commande), un `Index` (liste des commandes classées par ordre alphabétique avec une ligne d'entrée permettant de se déplacer facilement) et une recherche par mots clefs.

Si vous connaissez déjà le nom d'une commande et que vous désirez vérifier sa syntaxe (par exemple `factor`), vous pouvez saisir le début du nom de commande (disons `fact`) puis taper sur la touche de tabulation (située à gauche de la touche A sur un clavier français) ou cliquer sur le bouton `?` en haut à gauche. L'index des commandes apparaît alors dans une fenêtre, positionné à la première complétion possible, avec une aide succinte sur chaque commande. Par exemple, vous voulez factoriser un polynôme, vous supposez que le nom de commande commence par `fact`, vous tapez donc `fact` puis la touche de tabulation, vous sélectionnez à la souris `factor` (ou un des exemples) puis vous cliquez sur OK.

Vous pouvez aussi saisir `?factor` pour avoir l'aide succinte en réponse. Si le nom que vous avez saisi n'est pas reconnu, des commandes proches vous sont suggérées.

9 Les objets du calcul formel

9.1 Les nombres

Les nombres peuvent être exacts ou approchés. Les nombres exacts sont les constantes prédéfinies, les entiers, les fractions d'entiers et plus généralement toute expression ne contenant que des entiers et des constantes, comme `sqrt(2)*e^(i*pi/3)`. Les nombres approchés sont notés avec la notation scientifique standard : partie entière suivie du point de séparation et partie fractionnaire (éventuellement suivie de `e` et d'un exposant). Par exemple, `2` est un entier exact, `2.0` est la version approchée du même entier ; `1/2` est un rationnel, `0.5` est la version approchée du même rationnel. `Xcas` peut gérer des nombres entiers en précision arbitraire : essayez de taper `500!` et comptez le nombre de chiffres de la réponse.

On passe d'une valeur exacte à une valeur approchée par `evalf`, on transforme une valeur approchée en un rationnel exact par `exact` Les calculs sont effectués en mode exact si tous les nombres qui interviennent sont exacts. Ils sont effectués en mode approché si un des nombres est approché. Ainsi `1.5+1` renvoie un nombre approché alors que `3/2+1` renvoie un nombre exact.

```
sqrt(2)
evalf(sqrt(2))
sqrt(2)-evalf(sqrt(2))
exact(evalf(sqrt(2)))*10^9
exact(evalf(sqrt(2)*10^9))
```

Pour les nombres réels approchés, la précision par défaut est proche de 14 chiffres significatifs (la précision relative est de 53 ou 45 bits pour les réels flottants normalisés selon les versions de Xcas). Elle peut être augmentée, en donnant le nombre de décimales désiré comme second argument de `evalf`.

```
evalf(sqrt(2),50)
evalf(pi,100)
```

On peut aussi changer la précision par défaut pour tous les calculs en modifiant la variable `Digits`.

```
Digits:=50
evalf(pi)
evalf(exp(pi*sqrt(163)))
```

La lettre `i` est réservée à $\sqrt{-1}$ et ne peut être réaffectée ; en particulier on ne peut pas l'utiliser comme indice de boucle.

```
(1+2*i)^2
(1+2*i)/(1-2*i)
e^(i*pi/3)
```

`Xcas` distingue l'infini non signé `infinity` (∞), de `+infinity` ($+\infty$) et de `-infinity` ($-\infty$).

```
1/0; (1/0)^2; -(1/0)^2
```

Constantes prédéfinies	
`pi`	$\pi \simeq 3.14159265359$
`e`	$e \simeq 2.71828182846$
`i`	$i = \sqrt{-1}$
`infinity`	∞
`+infinity`	$+\infty$
`-infinity`	$-\infty$

9.2 Les caractères et les chaînes

Une chaîne est parenthésée par des guillemets (`"`). Un caractère est une chaîne ayant un seul élément.

```
s:="azertyuiop"
size(s)
s[0]+s[3]+s[size(s)-1]
concat(s[0],concat(s[3],s[size(s)-1]))
head(s)
tail(s)
mid(s,3,2)
l:=asc(s)
ss:=char(l)
string(123)
expr(123)
expr(0123)
```

Chaînes	
`asc`	chaîne->liste des codes ASCII
`char`	liste des codes ASCII->chaîne
`size`	nombre de caractères
`concat` ou `+`	concaténation
`mid`	morceau de chaîne
`head`	premier caractère
`tail`	chaîne sans le 1ier caractère
`string`	nombre ou expression->chaîne
`expr`	chaîne->nombre (base 10 ou 8) ou expression

9.3 Les variables

On dit qu'une variable est formelle si elle ne contient aucune valeur : toutes les variables sont formelles tant qu'elles n'ont pas été affectées (à une valeur). L'affectation est notée `:=`. Au début de la session `a` est formelle, elle devient affectée après l'instruction `a:=3`, `a` sera alors remplacé par 3 dans tous les calculs qui suivent, et `a+1` renverra 4. `Xcas` conserve tout le contenu de votre session. Si vous voulez que la variable `a` après l'avoir affectée, soit à nouveau une variable formelle, il faut la "vider" par `purge(a)`. Dans les exemples qui suivent, les variables utilisées sont supposées avoir été purgées avant chaque suite de commandes.

Il ne faut pas confondre

- le signe `:=` qui désigne l'affectation
- le signe `==` qui désigne une égalité booléenne : c'est une opération binaire qui retourne 1 ou 0 (1 pour true qui veut dire Vrai et 0 pour false qui veut dire Faux)
- le signe `=` utilisé pour définir une équation.

```
a==b
a:=b
a==b
solve(a=b,a)
solve(2*a=b+1,a)
```

On peut faire certains types d'hypothèses sur une variable avec la commande `assume`, par exemple `assume(a>2)`. Une hypothèse est une forme spéciale d'affectation, elle efface une éventuelle valeur précédemment affectée à la variable. Lors d'un calcul, la variable n'est pas remplacée mais l'hypothèse sera utilisée dans la mesure du possible, par exemple `abs(a)` renverra `a` si on fait l'hypothèse `a>2`.

```
sqrt(a^2)
assume(a<0)
sqrt(a^2)
assume(n,integer)
sin(n*pi)
```

La fonction `subst` permet de remplacer une variable dans une expression par un nombre ou une autre expression, sans affecter cette variable.

```
subst(a^2+1,a=1)
subst(a^2+1,a=sqrt(b-1))
a^2+1
```

Remarque : pour stocker une valeur dans une variable par référence, par exemple pour modifier une valeur dans une liste (un vecteur, une matrice), sans recréer une nouvelle liste mais en modifiant en place la liste existante, on utilise l'instruction `=<` au lieu de `:=`. Cette instruction est plus rapide que l'instruction `:=`, car elle économise le temps de copie de la liste.

9.4 Les expressions

9.4.1 Définition

Une expression est une combinaison de nombres et de variables reliés entre eux par des opérations : par exemple `x^2+2*x+c`.

Lorsqu'on valide une commande, `Xcas` remplace les variables par leur valeur si elles en ont une, et exécute les opérations.

```
(a-2)*x^2+a*x+1
a:=2
(a-2)*x^2+a*x+1
```

Certaines opérations de simplification sont exécutées automatiquement lors d'une évaluation :

- les opérations sur les entiers et sur les fractions, y compris la mise sous forme irréductible
- les simplifications triviales comme $x+0=x$, $x-x=0$, $x^1=x\ldots$
- quelques formes trigonométriques : $\cos(-x)=\cos(x)$, $\cos(\pi/4)=\sqrt{2}/2$, $\tan(\pi/4)=1\ldots$

Nous verrons dans la section **??** comment obtenir plus de simplifications.

9.4.2 Développer et simplifier une expression

En-dehors des règles de la section précédente, il n'y a pas de simplification systématique. Il y a deux raisons à cela. La première est que les simplifications non triviales sont parfois coûteuses en temps, et le choix d'en faire ou non est laissé à l'utilisateur ; la deuxième est qu'il y a en général plusieurs manières de simplifier une même expression, selon l'usage que l'on veut en faire. Les principales commandes pour transformer une expression sont les suivantes :

- `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` : d'un bon rapport temps d'exécution-simplification, elles 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`. Les deux ne tiennent pas compte des relations entre fonctions transcendantes (par exemple comme `sin` et `cos`).
- `factor` : un peu plus lente que les précédentes, elle écrit une fraction sous forme irréductible factorisée.
- `simplify` : elle essaie de se ramener à des variables algébriquement indépendantes avant d'appliquer `normal`. Ceci est plus coûteux en temps et "aveugle" (on ne contrôle pas les réécritures intermédiaires). Les simplifications faisant intervenir des extensions algébriques (par exemple des racines carrées) nécessitent parfois deux appels et/ou des hypothèses (`assume`) pour enlever des valeurs absolues avant d'obtenir la simplification souhaitée.
- `tsimplify` essaie de se ramener à des variables algébriquement indépendantes mais sans appliquer `normal` ensuite.

Dans le menu `Math` du bandeau supérieur, les 4 sous-menus de réécriture contiennent d'autres fonctions, pour des transformations plus ou moins spécialisées.

```
b:=sqrt(1-a^2)/sqrt(1-a)
ratnormal(b)
normal(b)
tsimplify(b)
simplify(b)
simplify(simplify(b))
assume(a<1)
simplify(b)
simplify(simplify(b))
```

La fonction `convert` permet de passer d'une expression à une autre équivalente, sous un format qui est spécifié par le deuxième argument.

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

Transformations	
`simplify`	simplifier
`tsimplify`	simplifier (moins puissant)
`normal`	forme normale
`ratnormal`	forme normale (moins puissant)
`expand`	développer
`factor`	factoriser
`assume`	rajout d'hypothèses
`convert`	transformer en un format spécifié

9.5 Les fonctions

9.5.1 Fonctions usuelles

De nombreuses fonctions sont déjà définies dans Xcas, en particulier les fonctions classiques. Les plus courantes figurent dans le tableau ci-après ; pour les autres, voir le menu Math.

Fonctions classiques	
abs	valeur absolue
round	arrondi
floor	partie entière (plus grand entier $\leq$)
ceil	plus petit entier $\geq$
abs	module
arg	argument
conj	conjugué
sqrt	racine carrée
exp	exponentielle
log	logarithme naturel
ln	logarithme naturel
log10	logarithme en base 10
sin	sinus
cos	cosinus
tan	tangente
asin	arc sinus
acos	arc cosinus
atan	arc tangente
sinh	sinus hyperbolique
cosh	cosinus hyperbolique
tanh	tangente hyperbolique
asinh	argument sinus hyperbolique
acosh	argument cosinus hyperbolique
atanh	argument tangente hyperbolique

9.5.2 Fonctions algébriques définies par l'utilisateur

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 `x^2-1`. 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)
f(2); f(a^2)
```

On peut définir des fonctions de plusieurs variables à valeurs dans $\mathbb{R}$ comme `f(x,y):=x+2*y` et des fonctions de plusieurs variables à valeurs dans $\mathbb{R}^p$ par exemple `f(x,y):=(x+2*y,x-y)`

9.5.3 Distinguer expression et fonction

Si `f` est une fonction d'une variable et `E` est une expression, `f(E)` est une autre expression. Il est essentiel de ne pas confondre fonction et expression. Si on définit : `E:=x^2-1`, alors la variable `E` contient l'expression x^2-1. Pour avoir la valeur de cette expression en $x=2$ il faut écrire `subst(E,x=2)` et non `E(2)` car `E` n'est pas une fonction. Lorsqu'on définit une fonction, le membre de droite de l'affectation n'est pas évalué. Ainsi l'écriture `E:=x^2-1; f(x):=E` définit la fonction $f : x \mapsto E$ car `E` n'est pas évalué. Par contre `E:= x^2-1; f:=unapply(E,x)` définit bien la fonction $f : x \mapsto x^2-1$ car `E` est évalué.

Le signe ' après une expression permet de calculer la dérivée d'une expression ou d'une fonction par rapport à la variable x. On l'utilisera si possible. Si on veut dériver par rapport à une autre variable ou plusieurs fois ou par rapport à plusieurs variables, on utilise la commande diff qui s'applique á une expression. 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
E'
diff(E)
f:=unapply(E,x)
f'
f'(2); f'(x)
diff(f(x))
f1:=function_diff(f)
```

Il ne **faut pas** définir la fonction dérivée par f1(x):=diff(f(x)), car x aurait dans cette définition deux sens incompatibles : c'est d'une part la variable formelle de dérivation et d'autre part l'argument de la fonction f1. D'autre part, cette définition évaluerait diff à chaque appel de la fonction (car le membre de droite d'une affectation n'est jamais évalué), ce qui serait inefficace. Il faut donc soit utiliser f1:=function_diff(f), soit f1:=unapply(diff(f(x)),x).

9.5.4 Opérations sur les fonctions

On peut ajouter et multiplier des fonctions, par exemple f:=sin*exp. Pour composer des fonctions, on utilise l'opérateur @ et pour composer plusieurs fois une fonction avec elle-même, on utilise l'opérateur @@.

```
f:=x->x^2-1
f1:=f@sin
f2:=f@f
f3:=f@@3
f1(a)
f2(a)
f3(a)
```

9.6 Listes, séquences, ensembles

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)

```
liste:=[1,2,4,2]
sequence:=(1,2,4,2)
ensemble:=%{1,2,4,2%}
```

Les listes peuvent contenir des listes (c'est le cas des matrices), alors que les séquences sont plates (un élément d'une séquence ne peut pas être une séquence). Dans un ensemble, l'ordre n'a pas d'importance et chaque objet est unique. Il existe une autre structure, appelée table, dont nous reparlerons plus loin.

Il suffit de mettre une séquence entre crochets pour en faire une liste ou entre accolades précédées de % pour en faire un ensemble. On passe d'une liste à sa séquence associée par op, d'une séquence à sa liste associée en la mettant entre crochets ou avec la fonction nop. Le nombre d'éléments d'une liste est donné par size (ou nops).

```
se:=(1,2,4,2)
li:=[se]
op(li)
nop(se)
```

```
nops(se)
%{se%}
size([se])
size(%{se%})
```

Pour fabriquer une liste ou une séquence, on utilise des commandes d'itération comme `$` ou `seq` (qui itèrent une expression) ou `makelist` (qui définit une liste à l'aide d'une fonction).

```
1$5
k^2 $ (k=-2..2)
seq(k^2,k=-2..2)
seq(k^2,k,-2..2)
[k^2$(k=-2..2)]
seq(k^2,k,-2,2)
seq(k^2,k,-2,2,2)
makelist(x->x^2,-2,2)
seq(k^2,k,-2,2,2)
makelist(x->x^2,-2,2,2)
```

La séquence vide est notée `NULL`, la liste vide `[]`. Pour ajouter un élément à une séquence il suffit d'écrire la séquence et 'élément séparés par une virgule. Pour ajouter un élément à une liste on utilise `append`. On accède à un élément d'une liste ou d'une séquence grâce à son indice mis entre crochets, le premier élément étant d'indice 0 (on peut aussi indicier en commencant à 1 en utilisant des parenthèses au lieu de crochets, mais attention au cas d'ambiguité avec la définition d'une fonction si on modifie un élément d'une liste)

```
se:=NULL; se:=se,k^2$(k=-2..2); se:=se,1
li:=[1,2]; (li:=append(li,k^2))$(k=-2..2)
li[0],li[1],li[2]
```

Les polynômes sont souvent définis par une expression, mais ils peuvent aussi être représentés par la liste de leurs coefficients par ordre de degré décroissant, avec comme délimiteurs `poly1[` et `]`. Il existe aussi une représentation pour les polynômes à plusieurs variables. Les fonctions `symb2poly` et `poly2symb` permettent de passer de la représentation expression à la représentation par liste et inversement, le deuxième argument détermine s'il s'agit de polynômes en une variable (on met le nom de la variable) ou de polynômes à plusieurs variables (on met la liste des variables).

Séquences et listes	
`E$(k=n..m)`	créer une séquence
`seq(E,k=n..m)`	créer une séquence
`[E$(k=n..m)]`	créer une liste
`makelist(f,k,n,m,p)`	créer une liste
`op(li)`	passer de liste à séquence
`nop(se)`	passer de séquence à liste
`nops(li)`	nombre d'éléments
`size(li)`	nombre d'éléments
`sum`	somme des éléments
`product`	produit des éléments
`cumSum`	sommes cumulées des éléments
`apply(f,li)`	appliquer une fonction à une liste
`map(li,f)`	appliquer une fonction à une liste
`poly2symb`	polynôme associé à une liste
`symb2poly`	coefficients d'un polynôme

9.7 Algèbre et analyse

Les instructions les plus souvent utilisées sont dans le menu `CAS` (calcul différentiel, simplification, arithmétique, etc.). Le menu `Cmds` contient d'autres instructions, en particulier tout ce qui est relatif à l'algèbre linéaire.

9.8 Instructions graphiques

Pour générer un graphe de fonction, surface, etc. vous pouvez utiliser les assistants du menu `Graphe`. Pour les suites récurrentes, créez auparavant un niveau de géométrie 2-d (menu Figure). Vous pouvez aussi créer des objets géométriques, en utilisant les instructions du menu `Geo`, par exemple `point(1,2)` affiche le point de coordonnées 1 et 2, `droite(A,B)` la droite passant par deux points A et B définis auparavant. On peut donner des attributs graphiques aux objets graphiques en ajoutant à la fin de l'instruction ayant un résultat graphique l'argument `affichage=...` dont la saisie est facilitée par le menu `Graphe->Attributs`.

Lorsqu'une ligne de commande contient une instruction graphique, le résultat est affiché dans un repère 2-d ou 3-d selon la nature de l'objet généré. On peut controler le repère avec les boutons situés à droite du graphique, par exemple orthonormaliser avec le bouton `_|_`. Si une ligne de commande contient des instructions graphiques et non graphiques, c'est la nature de la dernière instruction qui décide du type d'affichage.

L'instruction `A:=click()` permet de définir une variable contenant l'affixe d'un point du plan que l'on clique avec la souris.

9.9 Temps de calcul, place mémoire

Le principal problème du calcul formel est la complexité des calculs intermédiaires. Elle se traduit à la fois par le temps nécessaire à l'exécution des commandes et par la place mémoire requise. Les algorithmes implémentés dans les fonctions de `Xcas` sont performants, mais ils ne peuvent pas être optimaux dans tous les cas. La fonction `time` permet de connaître le temps d'exécution d'une commande (si ce temps est très court, `Xcas` exécute plusieurs fois la commande pour afficher un résultat plus précis). La mémoire utilisée apparaît dans les versions Unix dans la ligne d'état (en rouge à bas à gauche). Si le temps d'exécution d'une commande dépasse quelques secondes, il est possible que vous ayez commis une erreur de saisie. N'hésitez pas à interrompre l'exécution (bouton orange `stop` en bas à droite, il est conseillé de faire une sauvegarde de votre session auparavant).

10 Programmation

Comme le texte définissant un programme ne tient en général pas sur une ou deux lignes, il est commode d'utiliser un éditeur de programmes. Pour cela, on utilise le menu `Prg->Nouveau programme` de Xcas. Les boutons assistants `Fonctions, Test, Boucle` et le menu `Prg->Ajouter` facilitent la saisie des principales structures de controle de programmation.

10.1 Tests

On peut tester l'égalité de 2 expressions en utilisant l'instruction `==`, alors que `!=` teste si 2 expressions ne sont pas égales. On peut aussi tester l'ordre entre 2 expressions avec `<`, `<=`, `>`, `>=`, il s'agit de l'ordre habituel sur les réels pour des données numériques ou de l'ordre lexicographique pour les chaines de caractères.

Un test renvoie 1 s'il est vrai, 0 s'il est faux. On peut combiner le résultat de deux tests au moyen des opérateurs logiques `&&` (et logique), `||` (ou logique) et on peut calculer la négation logique d'un résultat de test `!` (négation logique). On utilise ensuite souvent la valeur du test pour exécuter une instruction conditionnelle `si ... alors ... sinon ... fsi`. N.B. : Xcas admet aussi une syntaxe compatible avec le langage C `if (condition) { bloc_vrai } else { bloc_faux }`.

Par exemple, on pourrait stocker la valeur absolue d'un réel x dans y par :

```
si x>0 alors y:=x; sinon y:=-x; fsi;
```

(on peut bien sur utiliser directement `y:=abs(x)`).

Exemples : Tester si un triangle dont on fait cliquer les 3 sommets à l'utilisateur est rectangle.

10.2 Boucles

On peut exécuter des instructions plusieurs fois de suite en utilisant une boucle définie (le nombre d'exécutions est fixé au début) ou indéfinie (le nombre d'exécutions n'est pas connu). On utilise en général une variable de controle (indice de boucle ou variable de terminaison).

– Boucle définie
  `for(init;condition;incrementation){ instructions }`
  `for ... from ... to ... do ... od`
  Exemple, calcul de 10!
  `f:=1; for (j:=1;j<=10;j++){ f:=f*j; }`
  `f:=1; for j from 1 to 10 do f:=f*j; od;`
  Attention, vous ne pouvez pas utiliser `i` comme indice de boucle, car `i` est prédéfini (comme $\sqrt{-1}$)
– Boucle indéfinie
  `while (...) { ... }`
  Exemple, algorithme d'Euclide
  `while (b!=0){ r:=irem(a,b); a:=b; b:=r;}`

Xcas accepte aussi l'arrêt de boucle en cours d'exécution (`if (...) break;`) dont l'usage peut éviter l'utilisation de variables de controle compliquées.

10.3 Fonctions (non algébriques)

La plupart des fonctions ne peuvent avoir une définition par une formule algébrique. On doit souvent calculer des données intermédiaires, faire des tests et des boucles. Il faut alors définir la fonction par une suite d'instructions, délimitées par `{ ... }`. La valeur calculée par la fonction est alors la valeur calculée par la dernière instruction ou peut être explicitée en utilisant le mot-clef `return` suivi de la valeur à renvoyer (N.B. : l'exécution de `return` met un terme à la fonction même s'il y a encore des instructions après).

Pour éviter que les données intermédiaires n'interfèrent avec les variables de la session principale, on utilise un type spécial de variables, les variables locales, dont la valeur ne peut être modifiée ou accédée qu'à l'intérieur de la fonction. On utilise à cet effet le mot-clef `local` suivi par les noms des variables locales séparés par des virgules. Si une fonction calcule plusieurs données on peut les renvoyer dans une liste.

Exemple : le PGCD

```
pgcd(a,b):={
  local r;
  while (b!=0){
    r:=irem(a,b);
    a:=b;
    b:=r;
  }
  return a;
}
```

On clique ensuite sur le bouton OK, si tout va bien, le programme `pgcd` est défini et on peut le tester dans une ligne de commande par exemple par `pgcd(25,15)`.

Dans la section suivante, on va voir comment exécuter en mode pas à pas un programme, ce qui peut servir à comprendre le déroulement d'un algorithme, mais aussi à corriger un programme erroné.

10.4 Exécuter en pas à pas et mettre au point

La commande `debug` permet de lancer un programme en mode d'exécution pas à pas. Elle ouvre une fenêtre permettant de diriger l'exécution du programme passé en argument. Par exemple, on entre le programme :

```
carres(n):={
  local j,k;
  k:=0;
  for (j:=1;j<n+1;j++) {
    k:=k+j^2;
  }
  return k;
}:;
```