Extraits du document : XCAS AU LYCEE
par Renée de Graeve et Bernard Parisse- Grenoble

Deuxième partie

Fiche Xcas calcul formel de base

L'exécution d'une ligne de commandes se fait par la touche "Entrée". Les nombres peuvent être exacts ou approchés (flottants). Les nombres exacts sont les constantes prédéfinies, les entiers, les fractions d'entiers et toutes expressions ne contenant que des entiers et des constantes. Les nombres approchés sont notés avec la notation scientifique standard : partie entière suivie du point de séparation et partie fractionnaire optionnellement suivie de `e` et d'un exposant.

Opérations	
`+`	addition
`-`	soustraction
`*`	mutiplication
`/`	division
`^`	puissance

Constantes prédéfinies	
`pi`	$\pi \simeq 3.14159265359$
`e`	$e \simeq 2.71828182846$
`i`	$i^2 = -1$ et `arg(i)`$=\frac{\pi}{2}$
`infinity`	∞
`+infinity` ou `inf`	$+\infty$
`-infinity` ou `-inf`	$-\infty$
`euler_gamma`	constante d'Euler

Fonctions classiques			
`evalf(t,n)`	évalue t avec n décimales	`sign`	signe (vaut -1, 0 ou +1)
`max`	maximum	`min`	minimum
`round`	arrondi	`frac`	partie fractionnaire
`floor`	plus grand entier $\leq$	`ceil`	plus petit entier $\geq$
`re`	partie réelle	`im`	partie imaginaire
`abs`	module ou valeur absolue	`arg`	argument
`conj`	conjugué	`affixe`	affixe
`factorial`	factorielle	`binomial`	coefficients binomiaux
`exp`	exponentielle	`sqrt`	racine carrée
`ln log`	logarithme naturel	`log10`	logarithme en base 10
`sin`	sinus	`cos`	cosinus
`tan`	tangente	`cot`	cotangente
`asin`	arc sinus	`acos`	arc cosinus
`atan`	arc tangente	`acot`	arc cotangente
`sinh`	sinus hyperbolique	`cosh`	cosinus hyperbolique
`asinh`	arc sinus hyperbolique	`acosh`	arc cosinus hyperbolique
`tanh`	tangente hyperbolique	`atanh`	arc tangente hyperbolique

Fractions	
`propfrac`	partie entière + partie fractionnaire
`getNum numer`	numérateur de la fraction simplifiée
`getDenom denom`	dénominateur de la fraction simplifiée
`f2nd`	`[numer,denom]` de la fraction simplifiée
`simp2`	simplification d'un couple
`mult_conjugue(n/d)`	multiplie par la quantité conjuguée de d ou de n si $d = 1$
`dfc`	développe un réel en fraction continue
`dfc2f`	transforme une fraction continue en réel

Chaînes de caractères, séquences et listes ou vecteurs	
`S:="abc"`	S est une chaîne de 3 caractères
`S:=a,b,c`	S est une séquence de 3 élements
`S:=[a,b,c]`	S est une liste de 3 élements
`op([a,b,c])`	renvoie la séquence `(a,b,c)`
`S:=""`	S est une chaîne de 0 caractère
`S:=NULL`	S est une séquence de 0 élement
`S:=[]`	S est une liste de 0 élement
`dim(S)`	renvoie le nombre d'éléments (ou caractères) de S
`S[0]`	renvoie le premier élément (ou caractère) de S
`S[n]`	renvoie le $n+1$ unième élément (ou caractère) de S
`S[dim(S)-1]`	renvoie le dernier élément (ou caractère) de S
`S:=S+"d"`	ajoute le caractère d à la fin de la chaîne S
`"ab"+"def"`	concaténe les 2 chaines et renvoie "abdef"
`S:=S,d`	ajoute l'élément `d` à la fin de la séquence S
`S:=append(S,d)`	ajoute l'élément `d` à la fin de la liste S

Fonctions d'arithmétique	
`a%p`	a modulo p est un élément de $\mathbb{Z}/p\mathbb{Z}$
`smod(a,b)` ou `(a%b)%0`	reste symétrique de la division euclidienne
`powmod(a,n,p)`	est l'entier égal à a^n modulo p
`irem`	reste de la division euclidienne
`iquo`	quotient de la division euclidienne
`iquorem`	quotient et reste de la division euclidienne
`ifactor`	décomposition en facteurs premiers
`ifactors`	liste des facteurs premiers
`idivis`	liste des diviseurs
`gcd`	plus grand diviseur commun
`lcm`	plus petit multiple commun
`iegcd`	identité de Bezout
`iabcuv`	renvoie $[u,v]$ tels que $au+bv=c$
`ichinrem`	restes chinois
`is_prime`	teste si l'entier est premier
`nextprime`	prochain entier pseudo-premier
`previousprime`	entier pseudo-premier précédent

Réécriture d'expressions			
`tsimplify`	simplifie (- puissant)	`simplify`	simplifie
`ratnormal`	forme normale (- puissant)	`normal`	forme normale
`partfrac`	décompose en éléments simples	`expand`	développe
`convert`	transforme en le format spécifié	`factor`	factorise
`pow2exp`	puissances vers exp	`expexpand`	développe les exp
`exp2pow`	convertit $\exp(n*\ln(x))$ en x^n	`lin`	linéarise les exp
`lncollect`	rassemble les log	`lnexpand`	développe les log
`trigsin`	utilise $\cos(x)^2=1-\sin(x)^2$	`halftan`	trig en $\tan(x/2)$
`tcollect`	linéarise et regroupe	`tlin`	linéarise
`texpand`	développe $\exp, \ln$ et trig	`trig2exp`	trig vers $\exp$
`hyp2exp`	hyperbolique vers $\exp$	`exp2trig`	$\exp$ vers trig

Quatrième partie

Fiche Xcas Algèbre

Polynômes	
normal	forme normale (développée et réduite)
expand	forme développée
ptayl(P,a)	polynôme de Taylor Q tel que P(x)=Q(x-a)
peval ou horner	évaluation en un point par l'algorithme de Horner
genpoly	polynôme défini par sa valeur en un point
canonical_form	trinôme mis sous forme canonique
coeff	coefficient ou liste des coefficients
poly2symb	du polynôme au format Xcas à la forme symbolique
symb2poly	de la forme symbolique à un polynôme au format Xcas
pcoeff	polynôme décrit par ses racines
degree	degré
lcoeff	coefficient du terme de plus haut degré
valuation	degré du monôme de plus bas degré
tcoeff	coefficient du monôme de plus bas degré
factor	décomposition en facteurs irréductibles sur $\mathbb{Q}$
cfactor	décomposition en facteurs irréductibles sur $\mathbb{Q}[i]$
factors	liste des facteurs irréductibles
divis	liste des diviseurs
collect	factorisation sur le corps des coefficients
froot	racines avec leurs multiplicités
proot	valeurs approchées des racines
sturmab	nombre de racines dans un intervalle
getNum	numérateur d'une fraction rationnelle non simplifiée
numer	numérateur d'une fraction rationnelle simplifiée
getDenom	dénominateur d'une fraction rationnelle non simplifiée
denom	dénominateur d'une fraction rationnelle simplifiée
propfrac	isole partie entière et fraction propre
partfrac	décomposition en éléments simples
quo	quotient de la division euclidienne
rem	reste de la division euclidienne
gcd	plus grand diviseur commun
lcm	plus petit multiple commun
egcd	identité de Bezout
chinrem	restes chinois
randpoly	polynôme aléatoire
cyclotomic	polynômes cyclotomiques
lagrange	polynômes de Lagrange
hermite	polynômes de Hermite
laguerre	polynômes de Laguerre
tchebyshev1	polynômes de Tchebyshev 1ère espèce
tchebyshev2	polynômes de Tchebyshev 2nde espèce

Pour les matrices et les chaînes de cararctères

Matrices	
`M:=[[a,b,c],[f,g,h]]`	`M` est une matrice de 2 lignes et 3 colonnes
`dim(M)`	est la liste [nbre_lignes, nbre_colonnes]
`nrows(M), ncols(M)`	nombre de lignes, colonnes
`M[0]`	renvoie la première ligne de `M`
`M[n]` ou `row(M,n)`	renvoie la $n+1$ unième ligne de `M`
`col(M,n)`	renvoie la $n+1$ unième colonne de `M`
`M[nrows(M)-1]`	renvoie la dernière ligne de `M`
`M[n..p]`	renvoie la sous matrice de `M` de lignes [n..p]
`M:=append(M,[d,k,l])`	ajoute la ligne `[d,k,l]` à la fin de `M`
`M[nrows(M)]:=[d,k,l]`	ajoute la ligne `[d,k,l]` à la fin de `M`
`M:=border(M,[d,k])`	ajoute la colonne `[d,k]` à la fin de `M`
`M[j,k]:=a`	modifie l'élément d'indice `j,k` en copiant `M`
`M[j,k]=<a`	modifie en place l'élément d'indice `j,k`

Opérations sur les vecteurs et matrices	
`v*w`	produit scalaire
`cross(v,w)`	produit vectoriel
`A*B`	produit matriciel
`A.*B`	produit terme à terme
`A^n`	puissance entière d'une matrice
`inv(A)` ou `A^-1`	inverse de la matrice `A`
`matpow(A,n)`	puissance symbolique d'une matrice
`tran(A)`	transposée de la matrice `A`
`trn(A)`	adjointe de la matrice `A`
`rank(A)`	rang de la matrice `A`
`trace(A)`	trace de la matrice `A`
`det(A)`	déterminant de la matrice `A`
`ker(A)`	base du noyau de la matrice `A`
`image(A)`	base de l'image de la matrice `A`
`idn(n)`	matrice identité de dimension `n`
`ranm(p,q)`	matrice `p*q` à coefficients aléatoires

Systèmes linéaires	
`linsolve`	résolution d'un système
`syst2mat`	transforme $Y = AX + b$ en la matrice A bordée par $-b$
`rref`	réduction de Gauss-Jordan
`ref`	réduction de Gauss
`rank`	rang du système
`det`	déterminant du système

Réduction des matrices	
`jordan`	diagonalisation ou réduction de Jordan
`pcar`	coefficients du polynôme caractéristique
`pmin`	coefficients du polynôme minimal
`companion`	matrice compagnon d'un polynôme unitaire
`eigenvals`	valeurs propres
`eigenvects`	vecteurs propres

Listes	
Créer une liste vide	L :=[ ]
Remplir une liste de six 0.	L :=[0$6] ou L :=[0$($k = 1..6$)] L :=makelist[0,$k = 1..6$]
Elément de rang k.	L[0] est le premier terme de la liste. L[k] est le terme de rang k.
Ajouter un élément x à la fin d'une liste ou une liste à la fin d'une liste.	append(L,x) Si L:=append(L1,L2) alors L:=[L1[0],L1[1],...L1[dim(L1)],L2]
Concaténer deux listes L1 et L2.	concat(L1,L2)
Dimension d'une liste.	dim (L)
Remplir une liste de 10 nombres aléatoires pris dans $\{1; 2; 3; 4; 5; 6\}$.	L :=[(1+rand(6))$($k = 1..6$)]

Fonctionnalités	Langage de programmation sous Xcas
Affecter "bonjour" à la variable MOT avec MOT une variable de type chaîne.	MOT := "bonjour" MOT[0]="b" et MOT[6]="r"
Longueur d'une chaîne.	dim(MOT)
Sous chaîne d'une chaîne.	mid(MOT,4,2) retourne la chaîne de longueur 2 située en $4^{ème}$ position, ici "ou". Attention le premier rang d'une chaîne est 0.
Concaténer des chaînes Avec PHRASE de type chaîne PHRASE←MOT + " papa"	PHRASE :=MOT+ " papa" PHRASE="bonjour papa"

Cinquième partie

Fiche Xcas pour la spécialité maths Terminale S.

Ces commandes se trouvent dans le menu Cmds.

Arithmétique	
`iquo(a,b), irem(a,b)`	quotient, reste de la division euclidienne de a par b
`isprime(p)`	Test de primalité
`gcd(a,b), lcm(a,b)`	PGCD, PPCM de 2 entiers
`iabcuv(a,b,c)`	Renvoie u et v tels que $au+bv=c$
`powmod(a,n,m)`	Renvoie $a^n \pmod m$
`L:=convert(a,base,b)`	liste des chiffres de a en base b. L[0]=ch des unités
`a:=convert(L,base,b)`	conversion inverse
`n:=a % b; p:=n % 0`	$n \in \mathbb{Z}/b\mathbb{Z}$ congru à a ; $p \in \mathbb{Z}$ et $p=a \pmod b$
Matrices et vecteurs	
`matrix(3,4,(j,k)->j+k)`	matrice définie par une formule
`M:=[[1,2,3],[4,5,6]]`	matrice définie par des coefficients (ou bien créer un tableur Xcas en lui donnant un nom de variable)
`v:=[0,1,0]`	vecteur défini par ses coordonnées
`M[0,1]` ou `M(1,2)`	élément de `M` ligne 1 colonne 2 : les indices commencent à 0 ou à 1 selon la notation
`M[j,k]:=a`	modifie l'élément d'indice j,k (copie de `M`)
`M[j,k]=<a` ou `M(j,k)=<a`	modifie en place l'élément d'indice j,k de `M`
`+,-,*`	addition, soustraction, produit de matrices/vecteurs
`inv(M)`	inverse d'une matrice carrée `M`
`M^n`	puissance entière d'une matrice `M`
`matpow(M,n)`	puissance (symbolique) d'une matrice `M`. Écrire `supposons(n>0)` si `M` n'est pas inversible
`P,D:=jordan(M)`	diagonalisation de la matrice `M` (hors programme)
Autres	
`asc("chaine")`	renvoie la liste des codes ASCII d'une chaine
`char(L)`	renvoie la chaine de caractères à partir d'une liste
`rsolve` et `seqsolve`	résolution de suites récurrentes
`L:=readrgb("fich")`	lecture du fichier image (jpg, png) dans `L`
`writergb("fich.png",L)`	stocke et affiche l'image contenue dans `L`
`rectangle(dx,0,dy/dx, gl_texture="fich")`	affiche l'image de taille `dx,dy` contenue dans le fichier "fich"

Attention : la notation `M(j,k):=a` est interprétée comme une définition de fonction si `j,k` est symbolique (non entier) et non comme une affectation de l'indice j,k de M. On peut utiliser la notation `M(j,k)=<a` qui modifie toutes les matrices partageant la représentation de M, est donc beaucoup plus rapide pour de grosses matrices, mais peut avoir des effets surprenants.

N.B. : il peut être nécessaire d'utiliser la version 0.9.9 ou ultérieure de Xcas.

Huitième partie

Fiche Xcas la programmation

1. **Pour écrire une fonction ou un programme**, il faut :
   - choisir une syntaxe, on décrit ici la syntaxe Xcas,
     - soit avec le menu Cfg▶Mode (syntax) ▶xcas,
     - soit en ouvrant la fenêtre de configuration du CAS en appuyant sur le bouton Config:... et choisir Xcas dans Prog style,
   - ouvrir un niveau éditeur de programme soit en tapant Alt+p, soit avec le menu Prg▶Nouveau programme. Une boite de dialogue s'ouvre pour faciliter la définition d'une nouvelle fonction.
   - Taper la fonction ou le script (suite de commandes) en utilisant les boutons Fonctions, Test et Boucles (assistant de création d'une fonction,d'un test ou d'une boucle) ou le menu Ajouter de l'éditeur. Le nom du programme et de ses arguments ne doit pas être un mot-clef ou une commande de Xcas, ces mots apparaissent en bleu et brun.
   - cliquer sur OK ou appuyer sur F9, pour compiler le programme (ou exécuter le script si on n'a pas terminé son écriture par :;).
   - pour exécuter le programme, il suffit de se placer dans une ligne de commandes vide, de taper le nom du programme suivi entre parenthèses par les valeurs des paramètres séparées par des virgules et d'appuyer sur Enter.

2. **Le menu Ajouter d'un niveau éditeur de programme**

Ce menu vous permet d'avoir la syntaxe d'une fonction, d'un test et des boucles.

Syntaxe d'une fonction :

```
f(x,y):={
  local z,a,...,val;
  instruction1;
  instruction2;
  val:=...;
  .....
  instructionk;
  retourne val;
};
```

Euclide et l'identité de Bézout :

```
Bezout(a,b):={
 local la,lb,lr,q;
  la:=[1,0,a];
  lb:=[0,1,b];
 tantque b!=0 faire
   q:=iquo(la[2],b)
   lr:=la+(-q)*lb;
   la:=lb; lb:=lr;
   b:=lb[2];
 ftantque
 retourne la;
};
```

3. **Compilation** La réponse à une compilation reussie sera Done si :; termine le programme et sera la traduction du programme si c'est ; qui le termine.
Pour Bezout(a,b), on clique sur OK (ou touche F9) et on obtient // Parsing Bezout // Success compiling Bezout puis Done. Puis, on tape : Bezout(78,56) et on obtient [-5,7,2] (car -5*78+7*56=2=pgcd(78,56)).

4. **Pas à pas** Vous pouvez exécuter un programme commandes par commandes ou le mettre au point grâce au débugueur. On tape : debug(Bezout(78,56))
Une fenêtre s'ouvre et on appuie sur sst pour une exécution au pas à pas.

Instructions en fançais	
affectation	`a:=2;`(a prend la valeur 2) et `purge(a);`(a redevient formelle)
entrée expression	`saisir("a=",a);`
entrée chaine	`saisir_chaine("a=",a);`
sortie	`afficher("a=",a);`
valeur retournée	`retourne a;`
arrêt de la boucle	`break;`
alternative	`si <condition> alors <inst> fsi;` `si <condition> alors <inst1> sinon <inst2> fsi;`
boucle pour	`pour j de a jusque b faire <inst> fpour;` `pour j de a jusque b pas p faire <inst> fpour;`
boucle répéter	`repeter <inst> jusqua <condition>;`
boucle tantque	`tantque <condition> faire <inst> ftantque;`
boucle faire	`faire<inst1>si(<condition>)break;<inst2>ffaire;`

Instructions comme en C++	
affectation	`a:=2;`(a prend la valeur 2) et `purge(a);`(a redevient formelle)
entrée expression	`input("a=",a);`
entrée chaine	`textinput("a=",a);`
sortie	`print("a=",a);`
valeur retournée	`return(a);`
arrêt de la boucle	`break;`
alternative	`if (<condition>) {<inst>};` `if (<condition>) {<inst1>} else {<inst2>};`
boucle pour	`for (j:= a;j<=b;j++) {<inst>};` `for (j:= a;j<=b;j:=j+p) {<inst>};`
boucle répéter	`repeat <inst> until <condition>;`
boucle tantque	`while (<condition>) {<inst>};`
boucle faire	`do <inst1> if (<condition>) break; <inst2> od;`
debug	lance le debugueur pour une exécution au pas à pas de l'argument

Signification des signes de ponctuation	
`.`	sépare la partie entière de la partie décimale
`,`	sépare les éléments d'une liste ou d'une séquence
`;`	termine chaque instruction d'un programme
`:;`	termine les instructions lorsqu'on ne veut pas l'affichage de la réponse
`!`	$n!$ est la factorielle de $n \in \mathbb{N}$ et $!(b)$ est l'inverse logique du booléen b

Opérateurs			
`+`	addition	`-`	soustraction
`*`	mutiplication	`/`	division
`^`	puissance	`a mod p`	a modulo p
`==`	teste l'égalité	`!=`	teste la différence
`<`	teste la stricte infériorité	`<=`	teste l'infériorité ou l'égalité
`>`	teste la stricte supériorité	`>=`	teste la supériorité ou l'égalité
`||, ou`	opérateur booléen infixé ou	`&&, et`	opérateur booléen infixé et
`non`	inverse logique de l'argument	`!(..)`	inverse logique de l'argument
`vrai`	est le booléen vrai ou true ou 1	`faux`	est le booléen faux ou false ou 0
`=`	permet de définir une équation	`:=`	affectation $(a := 2)$