Annexe 3

Guide d'utilisation pratique d'Axiom

CARPENT Quentin

CONIL Christophe

AC20: Utilisation de logiciels libres

Table des matières

Manipulation du logiciel

Démarrage d'Axiom

Axiom est un logiciel de calcul formel, originalement créé pour fonctionner en mode console. Cette utilisation se révèle pourtant peu pratique, et nécessite donc le lancement d'un programme graphique permettant de gérer une session Axiom. Dans une console, en environnement graphique, lancer TeX-macs par la commande:

da3@da3:~$ texmacs

Une fois celui-ci lancé changez les préférences pour avoir le logiciel en français: (Edit | Preferences | Language | French), puis insérer une zone de script Axiom : (Texte | Session | Axiome) Un carré bleu, avec une flèche rouge apparaît. La zone de frappe se situe après la flèche rouge.

Fermeture d'une session

Sous texmacs, utiliser simplement le bouton de fermeture. Attention, une demande de confirmation apparaît dans la barre d'état.

En mode console, utiliser «)quit » puis confirmez.

Instructions et commandes

Aide d'Axiom

On ne peut donner toutes les fonctions qui régissent le programme Axiom. Il est souvent nécessaire de rechercher une fonction grâce au système intégré. Pour cela, la démarche est la suivante:

1) On cherche par exemple à étudier une fonction complexe. On fait donc une recherche sur « complex ». Pour afficher tous les noms de fonction contenant ce terme, on utilise la commande:

(1) ->)what op complex

2) Une fois la série de fonctions affichées, il reste à choisir la fonction dont le nom se rapproche le plus de l'opération désirée. Utilisons le résultat complexSolve, qui, à première vue sert à résoudre des équations complexes. On utilise la commande

(1) ->)display op complexIntegrate

pour afficher les paramètres à fournir à la fonction. Il peut arriver qu'une fonction puisse prendre en compte plusieurs paramètres de type différents. Axiom se charge de choisir la fonction qui convient.

Il n'existe à l'heure actuelle pas d'explication « textuée » du fonctionnement fonctions.

ATTENTION: Axiom est un logiciel « case-sensitive », il fait attention à la casse des noms des fonctions entrées. Ainsi, complexIntegrate est différent de Complexintegrate (qui renverra une erreur).

Expressions Axiom

Il existe quantité d'expression Axiom: Affectations, fonctions, calculs directs...

Chacune de ces expression est interprétée par Axiom lorsqu'elle est écrite après son invite –> , et validée par la touche entrée. Un calcul simple se fait sans suffixe. Un calcul ne devant pas être affiché se terminera pas un point-virgule.

–> sin(%pi/3) va retourner $\frac{\sqrt{3}}{2}$

alors que

–> sin(%pi/3); Ne retournera rien

Assignation des variables et des fonctions

L'assignation des variables se fait par la commande :=

–> a:=7 enregistre 7 dans a, et transforme le type de a de variable vers un entier positif.

–>)clear properties a vide la variable a de son contenu, et détruit son type.

L'assignation des fonctions peut se fait par l'opérateur :=, mais cela n'est pas conseillé. En effet définir:

–> f:=3*x+4

fonctionne, même lors de la demande d'affichage de f(3), mais elle ne prends pas en compte l'ordre des variables. On utilisera plutôt la forme

f(x,y)==3*x+4*y

qui permet une définition correcte de la fonction (De plus la fonction sera compilée à l'exécution, et donc bien plus rapide).

ATTENTION: on ne peut pas, dans Axiom, faire abstraction, du signe * comme c'est le cas dans beaucoup de logiciels de calcul formel. 3x sera interprété comme 3(x), 3 étant une fonction. Il est donc nécessaire de toujours utiliser la syntaxe correcte: 3*x

Utilisation pratique d'Axiom

Plusieurs commandes sont pratiques dans l'utilisation d'Axiom:

% renvoie le résultat du dernier calcul. Ainsi,

–> %^-1 renvoie 1/(résultat précédent)

On utilise une autre commande, utilise pour transformer les types:

–> 3/7 renvoie 3/7, alors que

–> 3/7 :: Float renvoie 0.4285714285714285714

On pourra placer des commentaires, avec le préfixe - -

Commandes et raccourcis usuels :

Raccourci	Valeur
%pi	π
%plusInfinity	$+\infty$
%minusInfinity	$-\infty$
%e	e
%i	i

Récapitulatif de fonctions basiques

Étude de fonction

Les fonctions ici présentées peuvent être explicitées par l'ajout d'un paramètre. Par exemple, integrate(f(x)) est équivalent à integrate(f(x),x). On pourra ainsi modifier le second paramètre afin de satisfaire les besoins de l'exercice.

Utilisation de la fonction	Effet de la fonction
-> f(x)==x*(x + 3)	Affection de f(x).
-> f(5/3) :: Float	Donne un arrondi de f(5/3).
-> expand(f(x))	Développe les fonctions polynomiales.
-> factor(f(x))	Factorise les fonctions polynomiales.
-> integrate(f(x))	Intègre f(x) en fonction de la variable x
-> differentiate(f(x))	Calcule la dérivée de l'expression de f(x).
-> simplify(f(x))	Simplifie l'expression.
-> zeros(f(x))	Retourne les racines de la fonction.

Opérateurs élémentaires de calcul

Axiom connaît toutes les fonctions classiques dont les valeurs en x sont données par

exp(x), log(x), log10(x), round(x), sqrt(x), abs(x), floor(x)

mais aussi les fonctions trigonométriques:

sin(x), cos(x), tan(x), cot(x), sinh(x), cosh(x), tanh(x), coth(x)

et leurs fonctions inverses

asin(x), acos(x), atan(x), acot(x), asinh(x), acosh(x), atanh(x), acoth(x)

Fonctions de dénombrement : factorial(n), binomial(n,p)

Outre les opérateurs usuels d'égalité et d'inégalité = , <, <=, >, >=, Axiom reconnaît les opérateurs arithmétiques suivants:

Opérateur	Notation	Exemple
Addition	+	3+4=7
Soustraction	-	3-7=-4
Multiplication	*	3*7=21
Division	/	21/7=3
Puissance	** ou ^	2^10 ou 2**(2x+1) Note: En programmation, ^ ne fonctionne pas
Quotient	quo	7 quo 3 = 2
Reste	rem	7 rem 3 = 1
Négation Logique	~	if x ~= 3 then...

Type des variables dans Axiom

Différents types de nombres

Integer : Ensemble des entiers en base 10

PositiveInteger : Ensemble des entiers positif

NonNegativeInteger : Ensemble des entiers positifs ou nuls

Fraction Integer: Ensemble des Fractions

Float : Ensemble des réels à virgule

Notes:

- Si la fonction contient π, ou d'autres éléments spéciaux, elle prend comme type le nom de l'élément spécial. 3*%pi+5/7 est de type Pi.
- float(a,b) avec a et b 2 entiers, représente le nombre réel a*10^b
- Les fonctions spéciales, comme ln, sont du type de leur argument. log(7) sera de type Expression Integer.

Opérations sur les nombres

Comme dis précédemment, la conversion en réel se fait par :: Float. Cette syntaxe peut être utilisée en milieu de calcul:

–> sin(1::Float) sera de type Float et non de type Expression Integer. Le résultat renvoyé est donc manipulable, contrairement à sa version Entière.

–> factors(n) renvoie sous la forme d'une liste d'enregistrements la décomposition en facteurs premiers de n

–> gcd(a,b) renvoie le plus grand commun diviseur de a et b

–> lcm(a,b) retourne le plus petit commun multiple de a et b

–> prime?(a) retourne true ou false suivant que a est premier ou non.

Sommes

On utilisera la fonction sum, pour effectuer les sommes de séries.

sum(expression(k),k=a..b) retourne la somme des termes dépendants de k, k variant entre a et b inclus.

a et b sont des entiers positifs. Les bornes infinies ne sont pas applicables.

Exemples :

sum(k^2, k=1..100) retourne la somme des carrés entre 1 et 100

sum(k^2,k=1..n) retourne le terme général de la somme, pour tout n.

Produits

product(k^2,k=1..100) retourne le produit des carrés entre 1 et 100

Nombres Complexes

Les nombres complexes doivent être écris sous la forme a+b*%i

–> z:=a+b*%i	Définition de z
–> real(z)=a	Retourne la partie réelle de z
–> imag(z)=b	Retourne la partie complexe de z
–> argument(z)	Retourne l'argument de z
–> conjugate(z)=a-b*%i	Retourne le conjugué de z
-> z :: Complex Float	Arrondi les coefficients de z

Matrices

Définition d'une matrice

M := matrix [[1, 2, 3, 4] , [3, 5, 7, 9] , [6, 10, 14, 18]] crée la matrice $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 5 & 7 & 9 \\ 6 & 10 & 14 & 18 \end{bmatrix}$ et l'enregistre dans M.

Opérations courantes

–> Soient M1 et M2, 2 matrices supportant les opérations suivantes:

–> M1(3,4):=5 modifie la valeur de la 3eme ligne 4eme colonne, en lui affectant 5

–> M1+M2 retourne la somme des 2 matrices M1 et M2.

–> M1*M2 retourne le produit matriciel des 2 matrices.

–> M1^3 retourne le produit matriciel M1*M1*M1.

–> M1^-1 retourne la matrice inverse de M1, si celle ci est inversible.

–> trace(M1) retourne la trace de M1

–> determinant(M1) retourne le déterminant de M1

–> eigenvalues(M1) retourne les valeurs propres de M1

–> eigenvectors(M1) retourne les vecteurs propres associées de M1

–> characteristicPolynomial(M1,x) retourne le polynôme caractéristique de M1

Fonctions

Définition et fonctions usuelles

Comme vu précédemment, la définition d'une fonction se fait par :

–> f(x,y,z,t)==x**2+3*y+7*z+0*t

On peut à partir de cette fonction, utiliser diverses fonctions: en calculer les limites, faire des intégrations, calculer des dérivées...

Développement limité:

On pose tout d'abord le degré du développement à calculer (par exemple degré 4)

–>)set stream calculate 4

puis on cherche le développement:

-> series(f(x),x=0,4) donne le développement limité de f(x) à l'ordre 4 en x=0

–>)set stream calculate 5

-> series(f(x),x=%plusInfinity,5) donne le développement limité de f(x) à l'ordre 5 en x=$+\infty$

Fonctions définies par morceaux

On fait appel à des notions de programmation. On définit la fonction comme suit:

–> f(x) == if abs(x) < 1 then 1 else 1/x

La programmation est logique, on peut imbriquer des conditions. Les fonctions habituelles ne fonctionnent pas avec ces fonctions définies.

Résolution d'équations

Résolution d'équations

On utilise pour résoudre des équation la commande solve(expresssion, variable)

–> solve(x^4+1= 0, x) résout l'équation suivant la variable x

–> solve(sin(x)=0,x) résout l'équation suivant la variable x

On peut remarquer que cette dernière résolution d'équation ne donne que la solution 0.

Résolution de système d'équations

La résolution de système d'équation se fait aussi par l'intermédiaire de la fonction solve. Pour un système de la forme:

$$\begin{cases} a*x+b*y=c \\ d*b+e*y=f \end{cases}$$

On utilisera le système sans second membre associé

$$\begin{cases} a*x+b*y-c=0 \\ d*x+e*y-f=0 \end{cases}$$

puis la commande solve([expression1,expression2,..,expressionn],[x_1,x_2..x_n])

–> solve([a*x+b*y-c,d*b+e*y-f],[x,y])

La résolutions d'inéquations, ainsi que la résolution dans Z, n'est pas applicable.

Programmation

Attention: L'utilisation des boucles ci-dessous implique obligatoirement leur présence dans un fichier de commandes. L'utilisation directe en ligne de commande n'utilise pas la même structure, et est partiquement inutilisable.

Boucles for

L'utilisation de boucle for se fait grâce à la syntaxe

–> for *Variable* in *Intervalle* repeat *Instructions*

On peut ajouter un test en plus, dans une boucle for

–> for *Variable* in *Intervalle* | *test* repeat *Instructrions*

Exemples:

```
–>  for i in 1..10 repeat              - - Définition de la boucle
        ~prime?(i) => iterate          - - Test. Si i n'est pas premier, iterate ( recommencer la boucle)
        output(i)                      - - Afficher i

–>  for i in 1..10 | prime?(i) repeat
        output(i)
```

Ces 2 boucles renvoient les entiers premiers entre 1 et 10.

La commande iterate sert à revenir au début de la boucle, sans exécuter ce qui suit, la commande break sert à sortir de la boucle.

–> for w in ["Vive", "les", "boucles", "for!"] repeat output(w)

Cette boucle affichera chaque composant de la liste sur une ligne séparée. Attention, dans TeXmacs, ce résultat ne sera pas formaté.

Boucle while

La boucle while a exactement le même fonctionnement et la même structure que la boucle for.

–> while *Expression* repeat *Instructions*

Exemple

```
–> x := 1; y := 1
   while x <4 and y <10 repeat
        output [x,y]
        x := x + 1
        y := y + 2
```

Boucle repeat

La boucle repeat sert à répéter un nombre indéfini de fois une expression, jusqu'à ce qu'elle rencontre un break.

Exemple:

–> i := 1

repeat

if i>4 the break

output(i)

i := i+1

Programmation de fonctions

Il est possible dans Axiom de créer et de compiler une suite d'instructions logique, afin d'étudier par exemple une suite. Cette programmation ne se fait pas en général, par l'intermédiaire de la ligne de commande. La démarche est la suivante :

- Création d'un fichier texte vide, d'extension .input (par exemple exo1.input)
- Définition de la structure du programme
- Ecriture du code
- Chargement dans Axiom
- Execution

La structure de ce fichier texte est particulière. Etudions deux exemples:

Exemple 1:

Affiche les n premiers nombres premiers, et retourne le n-ième.

```
prime : Integer -> Integer          -- On définit le nom de la fonction (prime),
                                    -- suivi de  ':' puis du type de l'argument
                                    -- de '->' et enfin du type renvoyé

prime n ==                          -- L'argument passé à la fonction sera
                                    -- enregistré dans n
   i := 0                           -- On affecte différentes valeurs
   x := 0
   while not ( i = n ) repeat       -- Première boucle. Notons qu'on aurait pu
                                    -- écrire while i ~= n repeat
      while not prime?(x) repeat    -- On remarque que l'indentation est très
         x := x+1                   -- importante. C'est elle qui définit
                                    -- la présence dans la boucle ou non.
      print(x)
      i:=i+1                        -- On affiche la valeur, on augmente les variables
      x:=x+1
   x:=x-1
   x                                -- On retourne x ( valeur seule sur une ligne )
```

Exemple 2:

Définition de la somme de la suite $S_{2=} \sum v_n = \frac{1}{1} - (\frac{1}{2} + \frac{1}{4}) + \frac{1}{3} - (\frac{1}{6} + \frac{1}{8})....$

On remplace ici les espaces par des _, afin de bien voir l'indentation du programme

```
S2 : Integer -> Fraction Integer   -- Ici le type renvoyé est différent
                                   -- du type fourni.
S2 n ==
___if n = 1 then
_________1                         -- On retourne 1 si la condition est vraie.
______else                         -- Le else doit se situer indenté face au if.
_________if even?(n) then
_______________-(1/(2*n-2+1_       -- On utilise ici une stucture récursive
               /(2*n))+ S2 (n-1)
____________else
_______________1/n+S2 (n-1)
```

Une fois la structure du programme créée, il suffit d'enregistrer le travail dans le répertoire utilisateur, et d'utiliser la commande

–>)read exo1.input (à supposer que le nom de fichier soit exo1.input)

Puis de lancer la commande

–> S2 8

si on veut calculer la somme de la série de l'exemple 2 au rang 8.

L'utilisation de plusieurs variables est tout à fait possible, et se fait intuitivement.