χCAS pour TI Nspire CX et CX II

Bernard.Parisse@univ-grenoble-alpes.fr

2022

Table des matières

1 Introduction

Ce document explique comment prendre en main et utiliser efficacement sur calculatrices TI Nspire CX le système de calcul formel χCAS (une version adaptée du logiciel Xcas [1] pour cette calculatrice) ainsi que l'environnement MicroPython le plus complet à ce jour pour faire des mathématiques sur calculatrices (section 13).

χCAS est une calculatrice graphique formelle indépendante de la calculatrice nspire avec des fonctionnalités pour les élèves qui envisagent une poursuite d'études en sciences ou en maths : grande varitété de représentations graphiques, géométrie analytique, arithmétique et cryptographie (corps finis premiers et extensions, polynômes, etc.), algèbre, algèbre linéaire, analyse numérique, calcul flottant multi-précision et certifié (arithmétique d'intervalle), etc.

Toutes ces fonctionnalités sont intégrées, on peut représenter sur un même graphique un histogramme et un graphe de fonction et une droite, on peut écrire un programme pour faire une simulation et représenter les résultats graphiquement, ou utiliser les fonctions mathématiques dans un programme en syntaxe Python. De plus les

1. https://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html

sessions de calcul sont compatibles avec Xcas, Xcas pour Firefox et avec d'autres calculatrices compatibles χCAS (Numworks, Casio Graph 90 et 35eii).

Ici, on passera un peu de temps à apprendre comment utiliser le shell et les outils d'édition pour ensuite les faire travailler harmonieusement ensemble, sans autres limites que la mémoire et la vitesse de la TI (tout à fait raisonnable pour une calculatrice). Il est donc fortement recommandé de lire cette documentation pour une utilisation optimale de χcas, à l'exception de la section 18 qui s'adresse uniquement aux programmeurs qui souhaitent programmer leur calculatrice en C ou C++.

N.B. : Ce document est interactif, vous pouvez modifier les commandes et voir le résultat de l'exécution des commandes proposées en exemple en cliquant sur le bouton exe (ou en validant avec la touche Entrée).

2 Installation

Avant d'installer χCAS, vous devrez installer ndless [2] en suivant l'un des tutoriels suivants :

— Tutoriel si vous possédez une Nspire CX ou CX CAS avec OS 4.5.0 ou inférieur [3].

— Tutoriel si vous possédez une Nspire CX ou CX CAS avec un OS 4.5.x (x entre 1 et 3) [4]

— Tutoriel si vous possédez une Nspire CX II ou CX CAS II [5]

Ensuite, ouvrez le logiciel de communication de la TI Nspire et transférez

— `luagiac.luax.tns` et `khicaslua.tns` (interface pretty print)

— `khicas.tns` (interface shell)

— éventuellement le répertoire d'exemples `Xcas`.

2.1 Examens

— **Attention, certains concours ou examens interdisent l'utilisation de calculatrices formelles**. Il est de la responsabilité de l'utilisateur de vérifier que les calculatrices formelles sont autorisées avant d'utiliser χCAS dans un examen ou concours. Les auteurs ne sauraient être tenus pour responsables en cas d'utilisation non autorisée.

— χCAS est compatible avec le mode examen sur les Nspire 5.2 et 4.5.3 si on lance le mode examen depuis le shell de χCAS (cf. la section 4 pour lancer le shell, puis touche calculatrice entre esc et tab, puis touche e^x). Vous pouvez revenir de l'OS 5.3 vers l'OS 5.2 en utilisant le programme Backspire [6].

2. `https://ndless.me/`
3. https://tiplanet.org/forum/viewtopic.php?t=20446
4. https://tiplanet.org/forum/viewtopic.php?f=57&t=24267
5. https://tiplanet.org/forum/viewtopic.php?f=57&t=24264
6. https://tiplanet.org/forum/archives_voir.php?id=3152639

2.2 Mise en garde

Texas Instruments ne semble pas vouloir laisser ses utilisateurs libres d'utiliser des programmes ndless, et fait tout son possible pour rendre ndless inutilisable à chaque mise à jour (en général sous prétexte de "sécurité"). Si vous faites une mise à jour à partir d'octobre 2020, vous prenez le risque de ne plus pouvoir utiliser ndless donc de ne plus pouvoir utiliser χCAS. **Je vous conseille vivement de ne jamais faire de mise à jour de votre TI Nspire sans vous être bien renseigné sur le site tiplanet** [7].

3 Interface "pretty print"

Cette interface ouvre un écran de calcul similaire à celui du Scratchpad de la TI Nspire (A Calculs du menu principal). À privilégier si vous voulez surtout faire des calculs en profitant de la saisie 2d des expressions. Si vous n'êtes pas sur l'écran d'accueil de la Nspire, tapez la touche ON/HOME. Ensuite tapez 2, puis sélectionnez `khicaslua` dans le répertoire `ndless`.

Les commandes les plus utilisées de Xcas sont listées depuis la touche menu. On peut accéder au shell (section 4) en tapant `*` sur une ligne vide, et à l'éditeur de scripts (section 12) en tapant `+` ou `+"nom_de_fichier"` (mettre le nom du script sans l'extension `.py.tns`).

4 Interface shell : premiers pas

Si vous n'êtes pas sur l'écran d'accueil de la Nspire, tapez la touche ON/HOME. Ensuite tapez 2, puis sélectionnez `khicas` dans le répertoire `ndless`, et tapez sur enter (vous pouvez aussi aller dans le répertoire Xcas et sélectionner un exemple puis touche enter pour ouvrir un exemple). Ceci ouvre après une dizaine de secondes un "shell" dans lequel vous pourrez taper les commandes de calcul formel de Xcas. Lors de la première exécution vous devrez choisir entre l'interpréteur Xcas et l'interpréteur MicroPython. Taper enter pour choisir Xcas (sauf si vous êtes principalement intéressé par l'environnement Python, cf. la section 13).
Pour quitter χCAS et revenir à l'explorateur de la Nspire, il faut taper la touche doc (doc signifie documents) plusieurs fois (deux fois depuis le shell).

Par exemple, tapez `1/2+1/6` puis enter, vous devriez voir le résultat `2/3` s'afficher sur la ligne du dessous.

Vous pouvez recopier dans la ligne de commande une commande de l'historique en utilisant le curseur vers le haut ou vers le bas puis enter, puis vous pouvez modifier la commande et l'exécuter. Par exemple, taper sur la touche curseur vers le haut, enter et remplacez `1/6` par `1/3`.

Vous pouvez utiliser le résultat de la dernière commande avec la touche Ctrl-`Ans` de la calculatrice (ans en couleur au-dessus de la touche `(-)`). Il vaut en général mieux définir une variable comme résultat d'une commande si on souhaite la réutiliser. Pour cela, on utilise une des deux instructions d'affectation :

7. `https://tiplanet.org`

— l'affectation vers la droite => s'obtient avec la touche sto→ de la calculatrice (Ctrl puis var), par exemple 2=>A met 2 dans la variable A. Vous pouvez ensuite utiliser A dans un calcul, sa valeur sera remplacée par 2.

— l'affectation vers la gauche =. Par exemple A=2 fait la même chose que 2=>A.

Pour vous aider à saisir les commandes Xcas les plus utiles, χCAS dispose d'un catalogue d'une centaine de commandes, avec une courte description et le plus souvent un exemple d'exécution facile à recopier. Appuyez sur la touche doc, choisissez une catégorie avec le curseur, par exemple Algebre, tapez enter, puis choisissez une commande avec le curseur, par exemple factor. Un deuxième appui sur la touche menu vous affiche une courte description de la commande, en général avec un exemple. En tapant sur tab (ou enter), vous recopiez l'exemple en ligne de commande. Vous pouvez alors valider (enter) ou modifier la commande et valider (enter) pour factoriser un autre polynôme que celui donné en exemple.

Lorsqu'une commande renvoie une expression, celle-ci est affichée en écriture naturelle (affichage 2-d). Vous pouvez faire défiler l'affichage avec les touches du curseur lorsque l'expression est grande. Tapez sur esc pour revenir au shell.

Maintenant essayez de taper la commande plot(sin(x)). Indication : taper doc, puis sélectionner Courbes, ou shift-3.

Lorsqu'une commande renvoie un graphe, celui-ci est affiché. Vous pouvez modifier la fenêtre graphique d'affichage avec les touches + ou − (zoom in ou out), les touches du curseur, orthonormaliser le repère (touche /) ou faire une recherche automatique de l'échelle (autoscale touche *). Pour enlever ou remettre les axes et graduations, tapez sur var. Tapez sur esc pour revenir au shell.

Vous pouvez effacer l'historique des calculs et les variables pour commencer un nouvel exercice : depuis le menu doc sélectionnez 9 Effacer historique. Vous avez ensuite le choix entre effacer l'écran en conservant les variables (touche de validation à droite de la touche U) ou en les effaçant (esc). Vous pouvez visualiser la place occupée par les variables en tapant sur la touche var. Pour effacer une variable pour faire de la place en mémoire, sélectionnez la commande purge dans ce menu, puis taper le nom de variable à effacer (ou sélectionnez la variable depuis le menu var).

Pour quitter χCAS, appuyez sur la touche doc 2 fois. Lorsque vous lancez une autre application, les variables et l'historique des calculs sont sauvegardés (dans le fichier session.xw.tns du répertoire Xcas de la Nspire), ils seront restaurés lorsque vous reviendrez dans χCAS.

Remarques :

— Depuis le shell de calcul, les touches 1 à 9, 0, ., (et) précédées de shift font apparaitre un petit menu pour saisir rapidement certaines commandes.

— Lorsque le curseur est sur la ligne de commande juste après un nom de commande, l'appui sur la touche curseur vers le bas permet de voir l'aide sur la commande (si l'aide existe) et de saisir un exemple.

— Exemple : tapez shift 2, puis 3 (integrate), flèche vers le bas, puis tab ou enter
Modifiez l'expression à intégrer selon vos besoin puis tapez enter

5 Commandes usuelles de calcul formel

5.1 Développer et factoriser

Depuis le catalogue, sélectionner le sous-menu `Algebre` (2) ou le menu rapide shift-1

— `factor` : factorisation. Raccourci clavier shift-* (préfixé) ou =>⋆ (infixé touche sto puis *), par exemple `x^4-1=>⋆`

$$(x-1)(x+1)\left(x^2+1\right)$$

. Utiliser `cfactor` pour factoriser sur $\mathbb{C}$.

— `partfrac` : développement d'un polynôme ou décomposition en éléments simples pour une fraction. Raccourci clavier =>+ (touche sto puis +), par exemple `(x+1)^4=>+`

$$x^4+4x^3+6x^2+4x+1$$

ou `1/(x^4-1)=>+`

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

.

— `simplify` : essai de simplifier une expression. Raccourci clavier =>/ (touche → puis /), par exemple `sin(3x)/sin(x)=>/`

$$2\cos(2x)+1$$

Attention, cette commande est gourmande en mémoire, et la TI en a peu, le risque de reset existe.

— `ratnormal` : développer une expression, écrire une fraction sous forme irréductible.

5.2 Analyse

Depuis le catalogue (doc), sélectionner le sous-menu `Analyse` (4) ou le menu rapide shift-2

— `diff` : dérivation. On peut aussi utiliser la notation `'` (shift-*) pour dériver par rapport à x, ainsi
`diff(sin(x),x)`

$$\cos x$$

et `sin(x)'`

$$\cos x$$

sont équivalents. Pour dériver plusieurs fois, ajouter le nombre de dérivations par exemple `diff(sin(x^2),x,3)`

$$-8x^3\cos\left(x^2\right)-12x\sin\left(x^2\right)$$

.

— `integrate` : primitive si 1 ou 2 arguments, par exemple `integrate(sin(x))`

$$-\cos x$$

ou `integrate(1/(t^4-1),t)`

$$\frac{\ln|t-1|}{4}-\frac{\ln|t+1|}{4}-\frac{\arctan t}{2}$$

pour $\int \frac{1}{t^4-1}\,dt$
Calcul d'intégrale définie si 4 arguments, par exemple `integrate(sin(x)^4,x,0,pi)`

$$\frac{3}{8}\pi$$

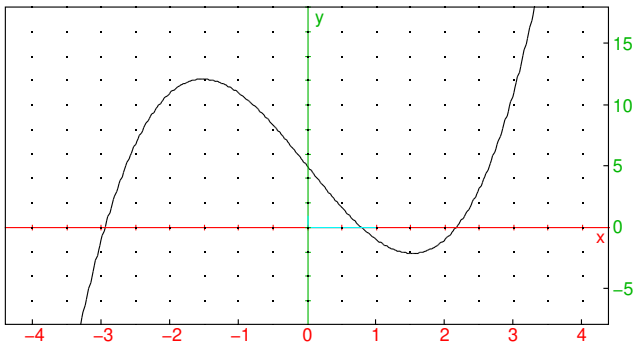
pour $\int_0^\pi \sin(x)^4\,dx$. Mettre une des bornes d'intégration sous forme approchée si on souhaite un calcul approché d'intégrale définie, par exemple `integrate(sin(x)^4,x,0.0,pi)`

$$1.1780972451$$

— `limit` : limite d'une expression. Exemple `limit((cos(x)-1)/x^2,x=0)`

$$-\frac{1}{2}$$

— `tabvar` : tableau de variations d'une expression. Par exemple `tabvar(x^3-7x+5)` on peut vérifier avec le graphe `plot(x^3-7x+5,x,-4,4)`



— `taylor` et `series` : développement de Taylor (ou développement limité ou asymptotique). Par exemple
`taylor(sin(x),x=0,5)`

$$x - \frac{x^3}{6} + \frac{x^5}{120} + x^6 \mathrm{order\_size}\,(x)$$

— `sum` : somme discrète. Par exemple
`sum(k^2,k,1,n)`

$$\frac{2\,(n+1)^3 - 3\,(n+1)^2 + n + 1}{6}$$

calcule $\sum_{k=1}^{n} k^2$,
`sum(k^2,k,1,n)=>*`

$$\frac{1}{6} n\,(n+1)\,(2n+1)$$

calcule la somme et l'écrit sous forme factorisée.

5.3 Résoudre

Depuis le catalogue, sélectionner le sous-menu `Resoudre` (menu puis touche ln)

— `solve` permet de résoudre de manière exacte une équation (se ramenant à une équation polynomiale). Il faut préciser la variable si ce n'est pas `x` par exemple `solve(t^2-1=0,t)`

$$[-1,1]$$

.
Si la recheche exacte échoue, la commande `fsolve` permet de faire une résolution approchée, soit par une méthode itérative en partant d'une valeur initiale `fsolve(cos(x)=x,x=0.0)`

$$0.739085133215$$

, soit par dichotomie `fsolve(cos(x)=x,x=0..1)`

$$[0.739085133215]$$

.
Pour avoir des solutions complexes, utiliser `csolve`.
On peut faire des hypothèses sur la variable que l'on cherche, par exemple `assume(m>1)`

$$m$$

puis `solve(m^2-4=0,m)`

$$[2]$$

.

— `solve` permet aussi de résoudre des systèmes polynomiaux simples, on donne en 1er argument la liste des équations, en 2ème argument la liste des variables. Par exemple intersection d'un cercle et d'une droite
`solve([x^2+y^2+2y=3,x+y=1],[x,y])`

$$[[0,1],[2,-1]]$$

— `linsolve` permet de résoudre des systèmes linéaires. On lui passe la liste des équations et la liste des variables (par convention une expression équivaut à l'équation expression=0). Par exemple
`linsolve([x+2y=3,x-y=7],[x,y])`

$$\left[\frac{17}{3},-\frac{4}{3}\right]$$

`linsolve` renvoie la solution générale du système (y compris si la solution n'est pas unique).

— `desolve` permet de résoudre de manière exacte certaines équations différentielles, par exemple pour résoudre $y' = 2y$, on tape `desolve(y'=2y)`.
Un exemple où on indique une condition initiale, la variable indépendante et la fonction inconnue :
`desolve([y'=2y,y(0)=1],x,y)` Utiliser `odesolve` pour une résolution approchée et `plotode` pour une représentation graphique de solution calculée de manière approchée.

— `rsolve` permet de résoudre de manière exacte certaines relations de récurrences $u_{n+1} = f(u_n, ...)$, par exemple les suites arithmético-géométriques, par exemple $u_{n+1} = 2u_n + 3, u_0 = 1$
`rsolve(u(n+1)=2*u(n)+3,u(n),u(0)=1)`

$$[4 \cdot 2^n - 3]$$

5.4 Arithmétique

Lorsque cela est nécessaire, on distingue l'arithmétique des entiers de celle des polynômes par l'existence du préfixe `i` (comme `integer`) dans un nom de commande, par exemple `ifactor` factorise un entier (pas trop grand) alors que `factor` factorise un polynôme (et `cfactor` factorise un polynôme sur les complexes). Certaines commandes fonctionnent à la fois pour les entiers et les polynômes, par exemple `gcd` et `lcm`.

5.4.1 Entiers

Depuis le catalogue, sélectionner le sous-menu `Arithmetic, Crypto` (menu 5)

— `iquo(a,b),irem(a,b)` quotient et reste de la division euclidienne de deux entiers.

```
iquo(23,13),irem(23,13)
```

$$1, 10$$

— `isprime(n)` teste si n est un nombre premier. Le test est probabiliste pour de grandes valeurs de n.
`isprime(2^64+1)`

faux

— `ifactor(n)` factorise un entier pas trop grand (jusque 128 bits environ). Par exemple
`ifactor(2^64+1)`

$$67280421310721 \cdot 274177$$

Raccourci clavier touches → puis * (=>$\star$)

— `gcd(a,b)`, `lcm(a,b)` PGCD et PPCM de deux entiers ou de deux polnômes

```
gcd(25,15),lcm(25,15)
```

$$5, 75$$

```
gcd(x^3-1,x^2-1),lcm(x^3-1,x^2-1)
```

$$x-1, \left(x^2+x+1\right)\left(x^2-1\right)$$

— `iegcd(a,b)` renvoie 3 entiers u, v, d tels que $au + bv = d$ où d est le PGCD de a et b, avec $|u| < |b|$ et $|v| < |a|$.

```
u,v,d:=iegcd(23,13); 23u+13v
```

$$[4, -7, 1], 1$$

— `ichinrem([a,m],[b,n])` lorsque cela est possible, renvoie c tel que $c = a \pmod m$ et $c = b \pmod n$ (si m et n sont premiers entre eux, c existe).

```
c,n:=ichinrem([1,23],[2,13]); irem(c,23);
 irem(c,13)
```

$$[93, 299], 1, 2$$

— `powmod(a,n,m)` calcule $a^n \pmod m$ par l'algorithme de la puissance rapide modulaire.

```
powmod(7,22,23)
```

$$1$$

Les commandes `asc` et `char` permettent de convertir une chaine de caractères en liste d'entiers (entre 0 et 255) et réciproquement, ce qui permet de faire facilement de la cryptographie avec des messages sous forme de chaines de caractères.

5.4.2 Polynômes

Depuis le catalogue, sélectionner le sous-menu `Polynomes` (8). La variable est par défaut x, sinon il faut la spécifier en général en dernier argument, par exemple `degree(x^2*y)` ou `degree(x^2*y,x)` renvoient 2, alors que `degree(x^2*y,y)` renvoie 1

— `coeff(P,n)` coefficient de x^n dans P, `lcoeff(P)` coefficient dominant de P, par exemple

```
P:=x^3+3x; coeff(P,1); lcoeff(P)
```

$$x^3 + 3x, 3, 1$$

— `degre(P)` degré du polynôme P.
`degree(x^3)`

3

— `quo(P,Q)`, `rem(P,Q)` quotient et reste de la division euclidienne de `P` par `Q`
`P:=x^3+7x-5; Q:=x^2+x; quo(P,Q); rem(P,Q)`

$$x^3+7x-5, x^2+x, x-1, 8x-5$$

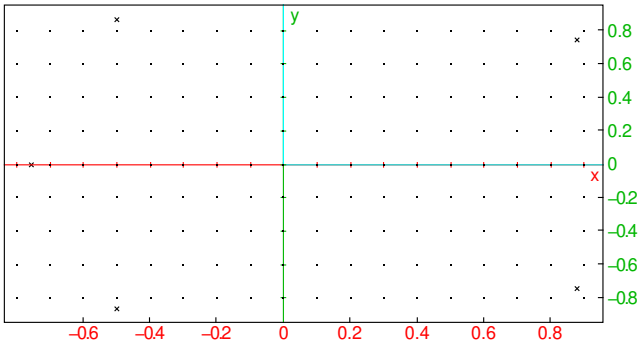
— `proot(P)` : racines approchées de P (réelles et complexes)

`proot(x^5+x+1)`

$[-0.754877666247, -0.5-0.866025403784\mathrm{i}, -0.5+0.866025403784\mathrm{i}, 0.877438833123-0.744861766\mathrm{62i}, 0$

Représentation graphique :

`point(proot(x^5+x+1))`



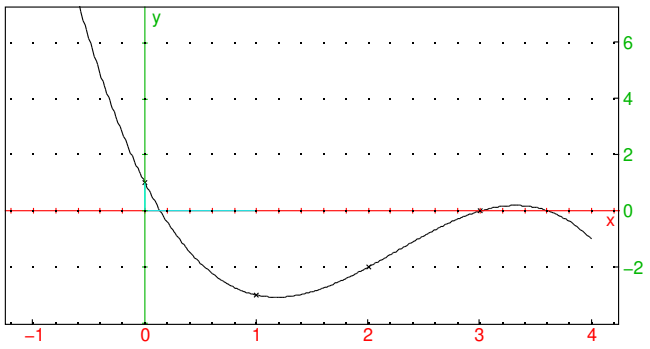
— `interp(X,Y)` : pour deux listes de même taille, polynôme d'interpolation passant par les points (X_i, Y_i).

```
X,Y:=[0,1,2,3],[1,-3,-2,0]; P:=interp(X,Y)=>+
```

$$[0,1,2,3]\,,[1,-3,-2,0]\,,\frac{-4x^3+27x^2-47x+6}{6}$$

Représentation graphique

```
scatterplot(X,Y); plot(P,x,-1,4)
```



— `resultant(P,Q)` : résultant des polynômes P et Q
`P:=x^3+7x-5; Q:=x^2+x; resultant(P,Q)`

$$x^3 + 7x - 5, x^2 + x, 65$$

— `hermite(x,n)` : n-ième polynôme de Hermite, orthogonal pour la densité $e^{-x^2}dx$ sur $\mathbb{R}$
— `laguerre(x,n,a)` : n-ième polynôme de Laguerre,
— `legendre(x,n)` : n-ième polynôme de Legendre, orthogonal pour la densité dx sur $[-1, 1]$
— `tchebyshev1(n)` et `tchebyshev2(n)` polynômes de Tchebyshev de 1ère et 2ème espèce définis par :

$$T_n(\cos(x)) = \cos(nx), \quad U_n(\cos(x))\sin(x) = \sin((n+1)x)$$

5.4.3 $\mathbb{Z}/n\mathbb{Z}$ et corps finis

Pour travailler avec des classes modulo n, utiliser la notation `a mod n`, par exemple `sqrt(2 mod 7)`. Ceci s'applique aussi pour travailler sur des corps finis premiers $\mathbb{Z}/p\mathbb{Z}$. Pour travailler sur des corps finis non premiers, il faut d'abord définir le corps avec `GF`, puis on utilise un polynôme en le générateur du corps.

5.5 Algèbre linéaire, vecteurs, matrices

Xcas ne fait pas de différence entre vecteur et liste. Par exemple pour faire le produit scalaire de deux vecteurs, on peut saisir :

```
v:=[1,2]; w:=[3,4]
```

```
dot(v,w)
```


Pour saisir une matrice élément par élément, taper sur shift-7 (touche M comme matrice) puis `matrix(` ou `doc i` (editer matrice). Vous pouvez ensuite créer une nouvelle matrice ou éditer une matrice existante parmi la liste de variables proposées. Pour de petites matrices, vous pouvez aussi entrer en ligne de commandes une liste de listes de même taille. Par exemple pour définir la matrice

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
A:=[[1,2],[3,4]]
```

ou `[[1,2],[3,4]]=>A`

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Il est fortement conseillé de stocker les matrices dans des variables pour éviter de les saisir plusieurs fois.

Pour entrer une matrice dont les coefficients sont donnés par une formule, on peut utiliser la commande `matrix`, par exemple

```
matrix(2,2,(j,k)->1/(j+k+1))
```

$$\begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{pmatrix}$$

renvoie la matrice dont le coefficient ligne j et colonne k vaut $\frac{1}{j+k+1}$ (attention les indices commencent à 0).

La matrice identité de taille n est renvoyée par la commande `idn(n)`, alors que `ranm(n,m,loi,[parametres])` renvoie une matrice à coefficients aléatoires de taille n, m. Par exemple

`U:=ranm(4,4,uniformd,0,1)`

$$\begin{pmatrix} 0.0881637986749 & 0.536643749103 & 0.360248812009 & 0.630750506185 \\ 0.0543795586564 & 0.819389336277 & 0.251521021593 & 0.0686232172884 \\ 0.178203014191 & 0.346399624366 & 0.852082391735 & 0.614078260958 \\ 0.714221911505 & 0.784336711746 & 0.453634891659 & 0.433968523517 \end{pmatrix}$$

`N:=ranm(4,4,normald,0,1)`

$$\begin{pmatrix} -0.57241508091 & -0.751538700822 & 0.989770363266 & -0.00575141421859 \\ 0.902209004052 & 0.455987131429 & -1.85967614286 & -0.243668831329 \\ -1.61539092837 & 0.558232005133 & -0.159613435564 & 1.35165685556 \\ -0.16425534864 & 1.27150836747 & 0.245690624229 & -0.595841236292 \end{pmatrix}$$

Pour exécuter une commande sur des matrices, s'il s'agit d'arithmétique de base (`+,-,*` inverse), on utilise les opérations au clavier. Pour les autres commandes. depuis le catalogue, sélectionner le sous-menu `Matrices` (menu sin)

— `eigenvals(A)`

$$\frac{\sqrt{33}+5}{2}, \frac{-\sqrt{33}+5}{2}$$

`eigenvects(A)`

$$\begin{bmatrix} \sqrt{33}-3 & -\sqrt{33}-3 \\ 6 & 6 \end{bmatrix}$$

renvoient les valeurs propres et vecteurs propres d'une matrice carrée A.

— `P,D:=jordan(A)`

$$\begin{bmatrix} \sqrt{33}-3 & -\sqrt{33}-3 \\ 6 & 6 \end{bmatrix}, \begin{bmatrix} \frac{\sqrt{33}+5}{2} & 0 \\ 0 & \frac{-\sqrt{33}+5}{2} \end{bmatrix}$$

calcule la forme normale de Jordan d'une matrice A (à coefficients exacts) et renvoie les matrices P et D telles que $P^{-1}AP = D$, avec D triangulaire supérieure (diagonale si A est diagonalisable)

— `Ak:=matpow(A,k)`

$$\begin{bmatrix} \frac{1}{66}\left(\sqrt{33}-3\right)\left(\frac{\sqrt{33}+5}{2}\right)^k\sqrt{33}-\frac{1}{66}\left(-\sqrt{33}-3\right)\left(\frac{-\sqrt{33}+5}{2}\right)^k\sqrt{33} & \frac{1}{132}\left(\sqrt{33}-3\right)\left(\frac{\sqrt{33}+5}{2}\right)^k\left(\sqrt{33}+11\right) \\ \frac{6}{66}\left(\frac{\sqrt{33}+5}{2}\right)^k\sqrt{33}-\frac{6}{66}\left(\frac{-\sqrt{33}+5}{2}\right)^k\sqrt{33} & \frac{6}{132}\left(\frac{\sqrt{33}+5}{2}\right)^k\left(\sqrt{33}+11\right) \end{bmatrix}$$

calcule la puissance k-ième d'une matrice A avec k une variable formelle.

— `rref` effectue la réduction sous forme échelonnée d'une matrice A (pivot de Gauss)

— `lu` calcule la décomposition LU d'une matrice A et renvoie une permutation de matrice P et deux matrices L triangulaire inférieure et U triangulaire supérieure telles que $PA = LU$. Le résultat de la commande

`P,L,U:=lu(A)`

$$[0,1],\begin{bmatrix}1&0\\3&1\end{bmatrix},\begin{bmatrix}1&2\\0&-2\end{bmatrix}$$

peut être passé en argument à la commande `linsolve(P,L,U,v)`

$$\left[0,\frac{1}{2}\right]$$

pour résoudre un système $Ax=b$ de matrice A en résolvant deux systèmes triangulaires (calcul en $O(n^2)$ au lieu de $O(n^3)$).

— `qr` calcule la décomposition QR d'une matrice A et renvoie deux matrices Q orthogonale et R triangulaire supérieure telles que $A=QR$.

— `svd(A)` calcule la factorisation en valeurs singulières d'une matrice A, et renvoie U orthogonale, S vecteur des valeurs singulières, Q orthogonale tels que `A=U*diag(S)*tran(Q)`. Le rapport de la plus grande valeur singulière de S sur la plus petite donne le nombre de condition de A relativement à la norme euclidienne, plus ce nombre est grand, plus on perd en précision en résolvant un système $Ax=b$ lorsque b n'est pas connu exactement.

6 Probabilités et statistiques

6.1 Tirages aléatoires

Depuis le catalogue, sélectionner le sous-menu `Probabilites` (menu 9) puis sélectionnez `rand()`

$$0.38078169385$$

(réel selon la loi uniforme dans $[0,1]$) ou

`n:=6:; randint(n)`

$$\text{“Done”},6$$

(entier entre 1 et n). De nombreuses autres fonctions aléatoires existent, avec comme préfixe `rand`, suivi par le nom de la loi, par exemple `randbinomial(n,p)` renvoie un entier aléatoire selon la loi binomiale de paramètres n,p. Pour créer un vecteur ou une matrice aléatoire, utiliser la commande `ranv` ou `ranm` (menu `Alglin, Matrice`), par exemple pour un vecteur de 10 composantes selon la loi normale centrée réduite

`ranv(10,normald,0,1)`

$$[-2.28261976775,0.652261860753,-0.690651824321,-0.323190436625,0.157460267236,-0.324617014178,-0.3$$

6.2 Lois de probabilités

Depuis le catalogue, sélectionner le sous-menu `Probabilites` (9). Les lois proposées dans le catalogue sont la loi binomiale, la loi normale, la loi exponentielle et la

loi uniforme. D'autres lois sont disponibles depuis `Tout` : `chisquared`, `geometric`, `multinomial` `studentd`, `fisherd`, `poisson`.

Pour obtenir la distribution cumulée d'une loi, on saisit le nom de la loi et le suffixe `_cdf` (sélectionner `cdf` dans le catalogue sous-menu Probabilités et taper F1). Pour obtenir la distribution cumulée inverse, on saisit le nom de la loi et le suffixe `_icdf` (sélectionner `cdf` dans le catalogue sous-menu Probabilités et taper F2).

Exemple : calcul de l'intervalle centré I pour la loi normale de moyenne 5000 et d'écart-type 200 tel que la probabilité d'être en-dehors de I soit de 5% :

```
M:=5000; S:=200; normald_icdf(M,S,0.025);normald_icdf
(M,S,0.975)
```

$$5000, 200, 4608.00720309, 5391.99279691$$

6.3 Statistiques descriptives 1-d

Ces fonctions agissent sur des listes

```
l:=[9,11,6,13,17,10]
```

Depuis le catalogue, sélectionner le sous-menu `Statistiques` (menu log)

— `mean(l)`

$$11$$

: moyenne arithmétique d'une liste

— `stddev(l)`

$$\frac{\sqrt{105}}{3}$$

: écart-type d'une liste
Utiliser
`stddevp(l)`

$$\sqrt{14}$$

pour avoir un estimateur non biaisé de l'écart-type d'une population dont `l` est un échantillon

— `median(l)`

$$10.0$$

, `quartile1(l)`

$$9.0$$

,
`quartile3(l)`

$$13.0$$

renvoient respectivement la médiane, le 1er et 3ème quartiles d'une liste
Pour les statistiques 1-d de listes avec effectifs, on remplace `l` par deux listes de même longueur, la 1ère liste est la liste des valeurs de la série statistique, la 2ème liste est la liste des effectifs. Voir aussi les commandes du menu `shift-3 histogram` et `barplot`.

6.4 Statistiques descriptives 2-d, régressions.

Entrez les deux listes de données dans deux variables, par exemple `X:=[1,2,3,4,5]` et `Y:=[3,5,6,8,11]`, ou dans une variable matrice ayant 2 colonnes, depuis le shell avec le menu `shift-6 8 matrix(`.
Depuis le catalogue, sélectionner le sous-menu `Statistiques` (touches `menu log`), pour les régressions, depuis le shell, tapez `shift-6`.

- `correlation(X,Y)` calcule la corrélation entre 2 listes de même taille.
- `covariance(X,Y)` calcule la covariance entre 2 listes de même taille.
- les commandes de suffixe `_regression(X,Y)` calculent des ajustements par régression au sens des moindres carrés, les commandes de suffixe `_regression_plot` tracent la courbe représentative de la régression (Ces commandes affichent de plus le coefficient R^2 qui permet de quantifier la qualité de l'ajustement (plus R^2 est proche de 1, meilleur est l'ajustement). Khicas a des commandes pour faire des régressions linéaires, exponentielles, logarithmiques, puissance, polynomiales et logistique.
- Par exemple `linear_regression(X,Y)` renvoie les coefficients m, p de la droite de régression linéaire $y = mx + p$.
  `linear_regression_plot(X,Y)` trace la droite d'ajustement des données contenues dans les listes `X,Y` de même taille.
- voir aussi les commandes `scatterplot`, `polygonplot` et `polygonscatterplot` pour afficher les données sur un graphique. On peut superposer plusieurs courbes de régression sur le même graphe en les séparant par un `;` en ligne de commande.

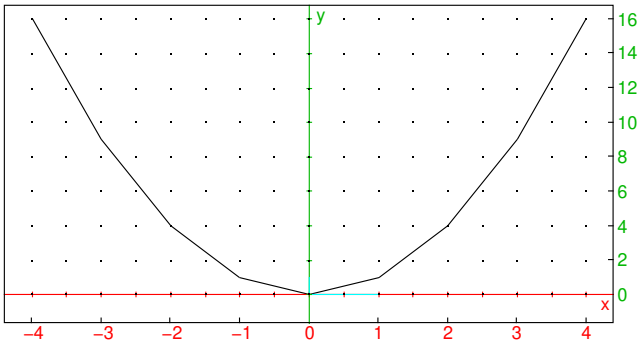
Ainsi pour afficher la droite de régression linéaire correspondant aux données `X=[1,2,3,4,5]` et `Y=[3,5,6,8,11]`, tapez les deux commandes ci-dessus ou éditez une matrice ayant 2 colonnes avec le raccourci `shift-6 8`. Puis `shift-6 enter` (ou `menu log` puis sélectionnez la commande `linear_regression_plot()` complétez la commande par `X,Y)` ou par le nom de variable de la matrice puis enter.

Remarque : si vos données sont dans une matrice `m` ayant 2 lignes au lieu de 2 colonnes, vous pouvez utiliser `m^*` pour transposer (en fait cela transconjugue, ce qui est identique pour des données réelles).

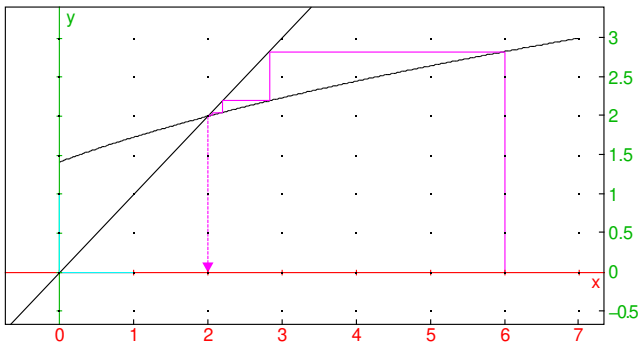
7 Courbes et autres représentations graphiques

Depuis le catalogue, sélectionner le sous-menu `Courbes` (accès rapide par menu 7, ou shift-3 depuis le shell).

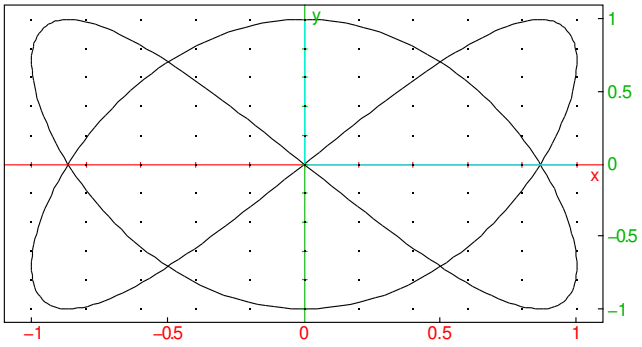
- `plot(f(x),x=a..b)` trace le graphe de $f(x)$ pour $x \in [a, b]$. On peut spécifier un pas de discrétisation avec `xstep=`, par exemple `plot(x^2,x=-4..4,xstep=1)`



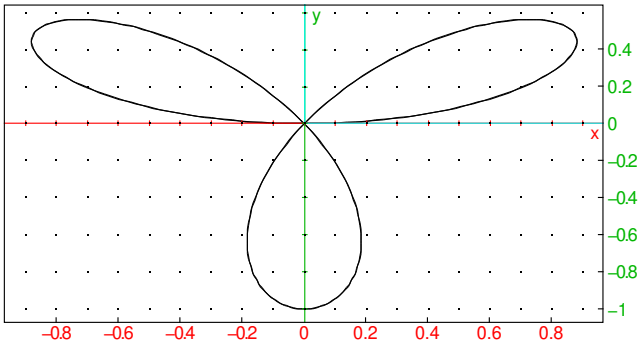
— `LineTan(f(x),x,x0)` trace la tangente au graphe de $f(x)$ en $x = x_0$.
— `plotarea(f(x),x=a..b,n,methode)` trace le graphe de $f(x)$ pour $x \in [a, b]$, et noircit une portion du plan qui approche l'aire sous la courbe, on peut préciser une méthode d'intégration avec `n` subdivisions parmi `rectangle_droit`, `rectangle_gauche`, `trapezes`, `simpson` (aller dans menu , Options puis taper le début du nom de la méthode pour la saisir plus rapidement)
— `plotseq(f(x),x=[u0,a,b])` graphe "en toile d'araignée" de la suite récurrente $u_{n+1} = f(u_n)$ de premier terme u_0 donné. Par exemple si $u_{n+1} = \sqrt{2 + u_n}, u_0 = 6$ avec une représentation sur $[0, 7]$
`plotseq(sqrt(2+x),x=[6,0,7])`



— `plotparam([x(t),y(t)],t=tm..tM)` courbe en paramétriques $(x(t), y(t))$ pour $t \in [t_m, t_M]$. On peut spécifier un pas de discrétisation avec `tstep=` par exemple
`plotparam([sin(2t),cos(3t)],t,0,2*pi)`



— `plotpolar(r(theta),theta=a..b)` courbe en polaires $r(\theta)$ pour $\theta \in [a,b]$, par exemple
`plotpolar(sin(3*theta),theta,0,2*pi)`

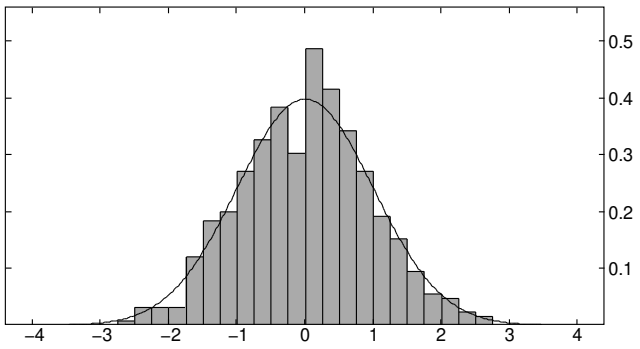


— `plotlist(l)` pour une liste `l`, trace la ligne polygonale reliant les points de coordonnées (i, l_i) (indice i commençant à 0).
`plotlist([X1,Y1],[X2,Y2],...)` trace la ligne polygonale reliant les points de coordonnées (X_i, Y_i)

— `scatterplot(X,Y)`, `polygonscatterplot(X,Y)` pour 2 listes `X,Y` de même taille, trace un nuage de points ou une ligne polygonale reliant les points de coordonnées (X_i, Y_i)

— `histogram(l,class_min,class_size)` trace l'histogramme des données de la liste `l` avec comme largeur de classe `class_size` en commençant à `class_min`. Par exemple, on peut tester la qualité du générateur aléatoire avec

```
l:=ranv(500,normald,0,1); histogram(l,-4,0.25
); plot(normald(x),x,-4,4)
```

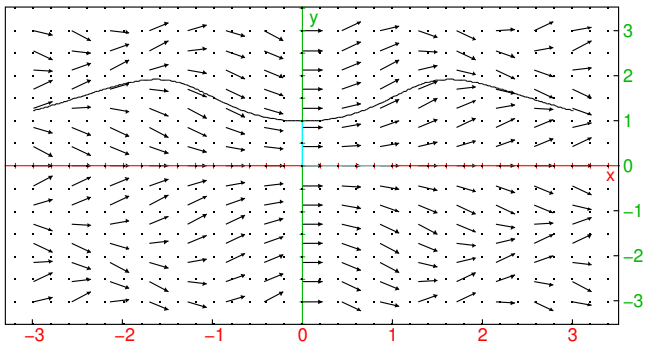


— `plotcontour(f(x,y),[x=xmin..xmax,y=ymin..ymax],[l0,l1,...])` trace les courbes de niveaux $f(x,y) = l_0, f(x,y) = l_1, ...$

— `plotdensity(f(x,y),[x=xmin..xmax,y=ymin..ymax])` représentation par niveaux de couleurs de la fonction de 2 variables `x` et `y` dans le rectangle spécifié (par défaut entre -4 et 4).

— `plotfield(f(t,y),[t=tmin..tmax,y=ymin..ymax])` trace le champ des tangentes à l'équation différentielle $y' = f(t,y)$. On peut ajouter en dernier paramètre optionnel `,plotode=[t0,y0]` pour tracer simultanément la solution passant par la condition initiale $y(t_0) = y_0$. Exemple $y' = \sin(ty)$ sur l'intervalle $[-3,3]$ en temps et $[-2,2]$ en y

```
plotfield(sin(t*y),[t=-3..3,y=-3..3],plotode=
[0,1])
```

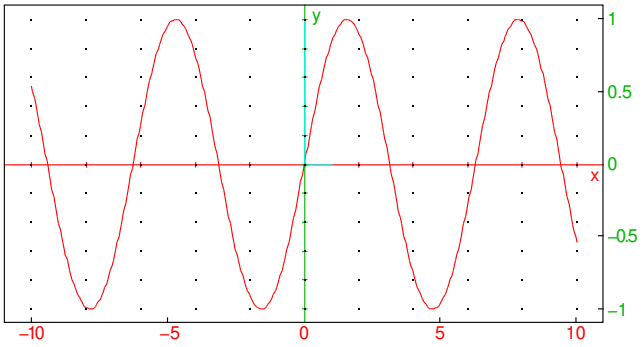


On peut aussi utiliser la commande `plotode` en-dehors d'une commande `plotfield`.

On peut tracer simultanément plusieurs graphiques, il suffit de séparer les commandes de tracé par `;`

Le menu `Options` (menu ,) vous permet de spécifier certaines options graphiques :

— pour les couleurs, on utilise `display=couleur`, par exemple `plot(sin(x),display=red)`

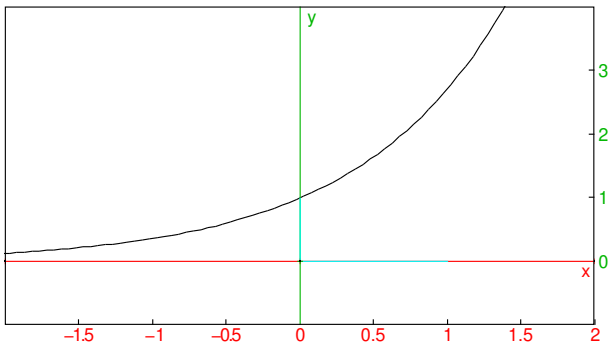


— Pour changer l'épaisseur des segments (y compris les lignes polygonales utilisées pour tracer une courbe), utiliser `display=line_width_2` à `display=line_width_8`. Pour changer à la fois la couleur et l'épaisseur, additionnez les attributs, par exemple : `display=red+line_width_2`

— Les cercles ainsi que les rectangles dont les bords sont parallèles aux axes peuvent être remplis avec l'attribut `display=filled` (qui peut s'additionner à d'autres attributs).

— Pour remplacer la fenêtre graphique calculée automatiquement par des valeurs prédéfinies, utiliser la touche OPTN, sélectionner `gl_x` ou/et `gl_y` et indiquer l'intervalle en x ou en y souhaité, la commande doit précéder une commande de tracé. Par exemple

```
gl_x=-2..2;gl_y=-1..4;plot(exp(x))
```



— Pour enlever les axes, sélectionner `axes` (`axes=0`). La commande doit précéder une commande de tracé.

Les versions récentes de χCAS permettent de représenter en 3d ou 4d des fonctions de 2 variables à valeur réelle ou complexe, par exemple `plot(x^2-y^2)` pour représenter la fonction de $\mathbb{R}^2 \to \mathbb{R}$ ou `plot((x+i*y)^2-9)` pour représenter la fonction de $\mathbb{C} \to \mathbb{C}$ qui à z associe $z^2 - 9$. Dans ce dernier cas, le module est représenté selon l'axe Oz et l'argument par une couleur de l'arc en ciel, de $-\pi$ en bleu violet à 0 en vert (en passant par jaune et orange) et de 0 à π en passant par cyan.

Pour préciser des options en 3d/4d, il faut utiliser la commande `plotfunc`, par exemple

```
plotfunc((x+i*y)^3-1,[x=-2..2,y=-2..2],nstep=500)
```

pour tracer $z \to z^3 - 1$ depuis le carré du plan complexe centré en l'origine de coté 4, avec une discrétisation utilisant 500 petits rectangles.

8 Géométrie analytique.

Les versions récentes de χCAS proposent une application de géométrie interactive 2d et 3d. L'application de géométrie permet de construire des figures dans le plan ou dans l'espace, et de faire bouger un point et tout ce qui en dépend pour illustrer certaines propriétés (géométrie dynamique). On peut faire des constructions de géométrie euclidienne pure, mais aussi avec des graphes de fonction, des coniques, etc. L'application possède deux "vues" : la vue graphique et la vue symbolique qui contient les commandes Xcas permettant de créer la figure (la philosophie de cette application est proche de celle du logiciel Geogebra, avec les commandes de Xcas).

Modes, vue graphique et symbolique.
Taper doc 1 pour afficher la liste des applications additionnelles, puis enter puis sélectionnez soit une nouvelle figure 2d ou 3d soit une figure existante. Vous pouvez aussi ouvrir l'application de géométrie depuis un graphe (par exemple après avoir tapé `plot(sin(x))` en tapant doc puis Sauvegarder figure.

Au lancement on est dans la vue graphique en mode repère, les touches de curseur permettent de changer de point de vue. Pour changer le mode, utiliser la touche menu, pour passer en vue symbolique et vice-versa taper enter. Par exemple tapez menu 3 pour passer en mode point qui permet de construire des points en déplaçant le pointeur et en tapant enter ou tapez menu 5 pour passer en mode triangle qui permet de construire un triangle à partir de ses 3 sommets, on déplace le pointeur et on tape enter trois fois. Pour déplacer le pointeur, utiliser les touches de déplacement, pour se déplacer plus rapidement, faire shift touche de curseur. Si on est proche d'un point existant, son nom apparait en bas. Pour déplacer le pointeur vers un point existant, vous pouvez aussi taper le nom du point (par exemple A).

Le mode pointeur permet de sélectionner un point et de le déplacer pour observer comment la construction varie, ce qui permet de mettre en évidence des propriétés de la figure, par exemple concurrence de 3 droites.

Si vous tapez sur la touche Back depuis la vue graphique de l'application, vous revenez au mode repère ou si vous y étiez vous passez en vue symbolique. Vous pouvez ajouter des objets à la construction depuis cette vue, en mettant une commande par ligne. Tapez ret pour passer à la ligne. Tapez enter pour revenir à la vue graphique. Dans la vue symbolique, vous pouvez sauvegarder la construction géométrique au format texte (avec une extension `.py`, même s'il ne s'agit pas d'un script Python). Tapez Back pour quitter l'application de géométrie.

Lorsque vous quittez l'application de géométrie, la figure est automatiquement sauvegardée dans une variable Xcas qui a le même nom que celui du nom de fichier affiché dans la vue symbolique. Vous pouvez purger la variable Xcas si vous voulez effacer la figure de la session.

Exemple : cercle circonscrit.
Depuis le shell, taper doc 1 sélectionner nouvelle figure 2d et valider enter. Puis menu 5 Triangle, enter pour créer le premier sommet du triangle puis déplacer le pointeur avec les touches de déplacement, enter pour créer le 2ème sommet du triangle, déplacer le pointeur à nouveau et enter pour créer le triangle.

Version longue en construisant le centre : Taper menu 7, sélectionner 8 Mediatrice, déplacer le pointeur de sorte que seul un segment du triangle soit sélectionné (affichage

en bas à droite `perpen_bisector D5,D`), taper enter pour créer la médiatrice du segment, déplacer le curseur sur une autre arête du triangle et enter pour créer la 2ème médiatrice, optionnellement sur le 3ème segment pour avoir les 3 médiatrices. Puis menu 6 et 4 Intersection unique. Déplacer le curseur vers une des médiatrices, taper enter puis vers une autre médiatrice, taper enter, ceci crée le centre du cercle circonscrit. Pour tracer le cercle, taper menu 4, déplacer le curseur au centre du cercle (vous pouvez utiliser les touches de déplacement ou juste taper H ou la bonne lettre si le centre du cercle s'appelle autrement), puis enter puis sur un des sommets et enter.

Version courte avec la commande `circonscrit` : taper menu 9 puis circonscrit puis sélectionner chaque sommet avec enter (A enter B enter C enter, remplacez si nécessaire A, B, C par les lettres du sommet du triangle).

Version en vue symbolique : taper Back puis en fin de script sur une ligne vide (taper ret s'il faut en créer une), taper
`c:=circonscrit(A,B,C)` enter

Exemple 3d : bac septembre 2019

Taper doc enter pour lancer l'application de géométrie puis nouvelle figure 3d. Puis Back ou enter pour passer en vue symbolique. Puis c = puis menu flèche haut deux fois pour sélectionner 3D puis enter puis 5 pour cube puis menu (aide), qui explique que les 2 premiers arguments de cube sont les sommets d'une arête, le troisième est un point d'un plan d'une face. Le premier exemple nous convient ici exactement, on tape Ans et on obtient `c=cube([0,0,0],[1,0,0],[0,1,0])` On tape enter pour voir le cube puis + plusieurs fois pour zoomer et enter pour revenir à la vue symbolique. Vous pouvez sauvegarder à tout moment la construction au format texte depuis le menu doc. On passe à la ligne en tapant shift enter. Puis on définit les sommets du cube en tapant `A,B,C,D,E,F,G,H:=` (taper A , B etc.), puis menu et flèche vers le haut 3 fois pour sélectionner Géometrie puis flèche vers le haut 4 fois pour sélectionner `sommets` enter et mettre c en argument `sommets(c)`. Taper enter pour visualiser puis enter à nouveau pour revenir en vue symbolique. Passer à la ligne avec ret puis créer le plan ABG en tapant P = puis shift 2 pour ouvrir le menu rapide lines et 8 pour saisir plane. La commande plan prend en arguments 3 points pour définir le plan (on peut aussi donner une équation cartésienne, ici A,B,G, `P=plan(A,B,G,` on va lui ajouter une couleur avec le menu rapide shift 4 disp `display=filled+green`, vérifier en visualisant avec enter enter. On passe à la ligne (ret) et on crée le segment DE S = shift 2 sélectionner la commande segment avec enter puis D,E et shift 4 pour lui donner une couleur `S=segment(D,E,color=cyan)` (on pouvait aussi créer le segment depuis la vue graphique en mode Lignes mais déplacer le pointeur est un peu lent). La construction est donc la suivante :

```
c=cube([0,0,0],[1,0,0],[0,1,0])
A,B,C,D,E,F,G,H=sommets(c)
P=plan(A,B,G,display=filled+green)
S=segment(D,E,display=cyan)
```

Vous pouvez taper enter pour la visualiser et utiliser les flèches de déplacement pour changer de point de vue. Taper enter ou Back pour revenir en vue symbolique. Pour quitter l'application taper Back. Taper F1 pour sauvegarder la figure si nécessaire. Vous pouvez depuis le shell de KhiCAS accéder à de nombreuses informations de

géométrie analytique, par exemple `equation(P)` (menu menu Géométrie) vous donnera l'équation cartésienne du plan P ou `is_orthogonal(P,S)` (menu Géométrie) vous confirmera que le plan P est orthogonal au segment S.

9 Unités et constantes physiques.

Le menu menu, constantes physiques (raccourci `menu pi`) et unités physiques (raccourci `menu sqrt`) affiche

— des commandes de gestion d'unité
`mksa` pour convertir vers les unités fondementales du système international
`usimplify` pour simplifier en utilisant une seule unité lorsque c'est possible, ou un produit de deux.
`ufactor` pour forcer l'écriture d'une unité en fonction d'une autre
`=>` (raccourci touche sto) pour convertir entre deux unités compatibles
— une liste non exhaustive d'unités physiques
— une liste de constantes physiques fondementales.

Exemples :

```
60_(km/h) => _(m/s)
mksa(_hbar_)
usimplify(1_W*1_s)
ufactor(1_W,1_J)
```

10 L'éditeur d'expressions

Lorsqu'un calcul renvoie une expression, elle est affichée en plein écran dans l'éditeur d'expression 2d. Depuis l'historique des calculs, si le niveau sélectionné est une expression, l'appui sur shift-5 (2d) affiche l'expression dans l'éditeur 2d. En ligne de commande, l'appui sur shift-5 ouvre aussi l'éditeur 2d, soit avec 0 si la ligne de commande était vide, ou avec le contenu de la ligne de commande si celle-ci est syntaxiquement correcte.

Lorsque l'éditeur 2d est ouvert, l'expression est affichée en plein écran et une partie de l'expression est sélectionnée. On peut alors agir sur la sélection en exécutant des commandes saisies via les menus ou le clavier, on peut aussi éditer la sélection (en mode de saisie 1d). Ceci permet de retravailler des sous-expressions ou d'éditer une expression en écriture naturelle.

Vous pouvez annuler la dernière modification effectuée en tapant sur shift-3 (undo).

Taper sur la touche enter pour quitter l'éditeur 2d et copier l'expression en ligne de commande, taper esc pour quitter sans recopier l'expression.

Exemple 1 : nous allons saisir

$$\lim_{x\to 0} \frac{\sin(x)}{x}$$

Depuis une ligne de commande vide, taper shift-5 (2d), vous devez voir 0 sélectionné. Tapez sur la touche x et enter, maintenant c'est x qui est en surbrillance. Tapez sur la

touche trig (trig renvoie sin, shift trig cos et ctrl trig tan), c'est $\sin(x)$ qui est en surbrillance. Tapez sur la touche de division (au-dessus de -), vous devez voir $\frac{\sin(x)}{0}$ avec 0 en surbrillance, tapez x puis enter, vous devez voir $\frac{\sin(x)}{x}$ avec x au dénominateur en surbrillance. Tapez sur le curseur flèche vers le haut pour mettre $\frac{\sin(x)}{x}$ en surbrillance, puis shift-2 4 (pour limit). L'expression est correcte, vous pouvez taper enter pour la recopier en ligne de commande et enter à nouveau pour exécuter le calcul. Si on avait voulu une limite en $+\infty$, il aurait fallu déplacer la sélection avec curseur vers la droite, puis faire shift-1 7 (oo) enter.

Exemple 2 : nous allons saisir

$$\int_0^{+\infty} \frac{1}{x^4+1}\,dx$$

Depuis une ligne de commande vide, taper shift-5 (2d), puis shift-2 3 (integrate), vous devez voir

$$\int_0^1 0\,dx$$

avec x sélectionné. Il faut donc changer le 1 de la borne supérieure et le 0 à intégrer. Pour modifier le 0, curseur vers la gauche pour le sélectionner puis `1/(x^4+1)` enter, puis curseur vers la gauche shift-1 7 enter. Taper sur enter pour recopier vers la ligne de commande puis enter pour effectuer le calcul, le résultat s'affiche dans l'éditeur 2d, enter quitte l'éditeur avec dans l'historique l'intégrale et sa valeur (en syntaxe algébrique 1d).

Exemple 3 : nous allons calculer et simplifier

$$\int \frac{1}{x^4+1}\,dx$$

Depuis une ligne de commande vide, taper shift-5 (2d), puis shift-2 3 (integrate), vous devez voir

$$\int_0^1 0\,dx$$

Déplacez le curseur sur le 0 de la borne inférieure de l'intégrale et tapez sur la touche DEL, vous devez voir

$$\int 0\,dx$$

avec le tout sélectionné. Utilisez le curseur vers le bas pour sélectionner 0 et tapez `1/(x^4+1)` enter puis enter pour recopier en ligne de commande puis enter pour exécuter le calcul, le résultat s'affiche maintenant dans l'éditeur 2d.
On peut alors sélectionner avec les touches du curseur par exemple l'argument d'un des arctangentes et exécuter shift-1 enter (simplify) pour effectuer une simplification partielle du résultat, puis recommencer avec l'autre arctangente.
On peut simplifier encore plus, en rassemblant les logarithmes. Pour cela il faut d'abord échanger deux des arguments de la somme. Sélectionnez un des logarithmes avec des déplacements du curseur, puis tapez shift-curseur droit ou gauche, cela échange l'argument sélectionné avec son frère de droite ou de gauche. Tapez ensuite ctrl curseur vers

la droite ou vers la gauche, ceci augmente la sélection en ajoutant le frère de droite ou de gauche. Une fois les deux logarithmes sélectionnés, taper shift-1 2 enter (factor), puis descendez la sélection sur la somme ou différence de logarithmes, allez dans le menu `menu` puis enter (Tout), tapez les lettres l, n, c ce qui déplace à la première commande commençant par `lnc`, sélectionnez `lncollect`, validez et tapez enfin sur enter (eval).

11 Sessions de calculs

11.1 Edition de l'historique.

En utilisant la touche curseur vers le haut/bas, on se déplace dans l'historique des calculs, le niveau courant est en surbrillance.

Pour modifier l'ordre des niveaux dans l'historique des calculs, tapez ctrl curseur vers le haut ou vers le bas. Pour effacer un niveau, appuyez sur la touche DEL (le niveau est recopié dans le presse-papiers).

Pour modifier un niveau existant, on tape sur shift-5 ou sur shift-4. Dans le premier cas, c'est l'éditeur 2d qui est appelé si le niveau est une expression, dans le deuxième cas, c'est l'éditeur texte qui est appelé. Taper esc pour annuler les modifications ou enter pour valider. Si les modifications sont validées, les lignes de commande situées en-dessous de la ligne modifiée seront automatiquement calculées, tenant compte des modifications, par exemple si vous modifiez un niveau comme `A:=1`, les lignes situées en-dessous dépendant de `A` seront actualisées.

Ce processus peut être automatisé en utilisant un curseur, que l'on peut créer avec un assistant, depuis le menu `doc`, Parameter. Une fois créé, vous pouvez modifier un curseur en tapant sur les touches + ou - lorsque le niveau contenant la commande `assume` ou `parameter` est sélectionné (tapez * ou / pour une modification plus rapide).

11.2 Variables

En appuyant sur la touche `var` vous affichez la liste des variables qui ont une valeur, ainsi que des commandes de gestion de variables. Déplacez le curseur vers une variable puis enter pour la recopier en ligne de commande, DEL copie en ligne de commande la commande d'effacement de la variable (confirmez ensuite avec enter). La commande `restart` permet d'effacer toutes les variables. La commande `assume` permet de faire une hypothèse sur une variable, par exemple `assume(x>5)` (pour saisir >, taper ctrl +)

11.3 Sauvegarde et compatibilité

Vous pouvez sauvegarder une session en tapant sur ctrl save ou doc 2. Vous pouvez ouvrir une session en tapant sur shift save ou doc 4. Les sessions sont sauvegardées avec un nom de fichier terminant par `.xw.tns`. Vous pouvez les ouvrir sur votre ordinateur avec Xcas ou Xcas pour Firefox. Réciproquement, doc Fich, Exporter comme

permet de sauvegarder une session dans ce format, ou dans un autre format pour utiliser votre session sur une calculatrice χCAS-compatible. Dans Xcas pour Firefox, il faut sélectionner dans les paramètres Nspire CX comme calculatrice puis exporter.

N.B. : vous pouvez sauvegarder le contenu de l'éditeur de script indépendamment de la session, les scripts ont un nom de fichier terminant par `.py.tns`. Il suffit de renommer le script en enlevant le `.tns` pour utiliser ce script avec n'importe quel interpréteur compatible Python, par exemple une calculatrice d'un autre constructeur proposant un interpréteur MicroPython. Réciproquement, vous pouvez importer un script Python en faisant une copie du fichier en ajoutant `.tns` au nom de fichier et en le transférant sur le TI Nspire CX. Vous pouvez alors l'ouvrir dans l'éditeur de script.

12 Programmation

L'environnement de programmation de χcas est assez complet : un éditeur, l'interpréteur de Xcas avec toutes ses commandes (compatibilité partielle avec les modules Python math, cmath, random, turtle, numpy, scipy, giacpy, matplotlib et un surensemble du module kandinsky), avec un outil de mise au point permttant l'exécution en pas à pas. Ce même environnement peut utiliser l'interpréteur MicroPython au lieu de Xcas (cf. section 13).

Vous pouvez programmer en utilisant les structures de commande en français de Xcas ou en utilisant la compatibilité de syntaxe Python. Les programmes très courts (en une ligne) peuvent être saisis directement en ligne de commande. Les programmes plus longs ou que l'on souhaite sauvegarder seront saisis dans l'éditeur de programmes sur la calculatrice, ou bien transférés depuis un PC ou une autre calculatrice.

Des programmes sont installés dans le répertoire Xcas. Vous pouvez les visualiser depuis le shell en tapant `doc` 4 (Charger), et les exécuter depuis le shell en tapant ctrl r.

12.1 Prise en main (programmation)

Un premier exemple en ligne de commande :
une fonction définie par une expression algébrique. On saisit `nom_fonction(parametres):=expression`
Par exemple, pour définir le périmètre d'un cercle de rayon r, on peut taper

```
peri(r):=2*pi*r
```

$$r \mapsto 2\pi r$$

(pour taper `:`, faire ctrl puis `,`) puis on peut calculer `peri(1)`

$$2\pi$$

.

Autre exemple, pour calculer l'intervalle de confiance de seconde connaissant une fréquence p et un effectif n, on tape

```
F(P,N):=[P-1/sqrt(N),P+1/sqrt(N)]
```

$$(P,N) \mapsto \left[\begin{array}{cc} P-\frac{1}{\sqrt{N}} & P+\frac{1}{\sqrt{N}} \end{array} \right]$$

puis on teste `F(0.4,30)`

$$[0.217425814165, 0.582574185835]$$

Autre exemple : avec la tortue de Xcas

La tortue de Xcas est un petit robot qui se déplace selon des ordres qui lui sont donnés en laissant une trace de son passage. Les commandes de la tortue sont accessibles depuis le dernier item du menu `menu`, par le raccourci x^2. Saisir la commande `avance` dans ce menu puis valider, vous devez voir la tortue (symbolisée par un triangle) avancer de 10 pixels. Taper esc pour revenir en ligne de commande. Saisir la commande `tourne_gauche` et valider, la tortue a tourné de 90 degrés. Répéter 3 fois ces deux commandes pour afficher un carré.

Pour effacer le dessin et ramener la tortue à l'origine, saisir la commande `efface`. Pour faire des dessins tortue, il est conseillé d'utiliser l'éditeur de programmes (cf. ci-dessous).

Autre exemple : une boucle "oneliner" en syntaxe Xcas.

Ouvrez le menu Programmes (menu cos), puis sélectionnez l'exemple de `pour` (curseur sur pour puis Ans)

```
pour j de 1 jusque 10 faire print(j,j^2); fpour;
```

$$0$$

tapez sur enter, vous devez voir les carrés des entiers de 1 à 10.

Exercice : faire faire un carré à la tortue en utilisant une boucle.

Utilisation de l'éditeur

Modifions cet exemple pour faire afficher les carrés de 1 à n en utilisant la syntaxe compatible Python et l'éditeur de programmes. Tapez sur esc pour passer du shell à l'éditeur de programmes. Si on vous demande programme ou Tortue faites enter. Ceci ouvre l'éditeur avec une maquette de fonction `def f(x):` Vérifiez que la syntaxe Python est activée (menu `doc`), sinon activez-la (8). Remplacez `x` par `n`, puis déplacez le curseur en fin de ligne et passez à la ligne (touche à la droite de la touche U). Tapez Shift-2 puis enter (`for`), placez un `j` entre `for` et `in range(` puis un `n` entre les parenthèses de `range()`. À la ligne suivante, tapez shift-3 7 (`print`) et validez (enter), puis tapez `j,j^2)`. Vous devriez avoir le texte source suivant :

```
def f(n):
  for j in range(1,n+1):
    print(j,j^2)
  return x
```

Remplacez `x` par `n` dans `return` (ou effacez la ligne). N.B. : pour la puissance, on peut utiliser `^` ou `**` dans KhiCAS (il faut utiliser `**` en Python).

Maintenant, tapez enter. Si tout va bien, vous devez voir `Success` dans la ligne d'état. Sinon, le numéro de ligne de la première erreur est indiqué ainsi que le mot qui a provoqué l'erreur. Le curseur est positionné sur la ligne où l'erreur a été détectée (il peut arriver que l'erreur soit située avant mais détectée un peu plus loin seulement). Si vous utilisez la syntaxe en Python, notez que les structures de programmation sont traduites en langage Xcas, les erreurs affichées le sont par rapport à cette traduction (donc des mots-clefs de fin de structure comme `end` peuvent avoir été ajoutées).

Si le programme est syntaxiquement correct, vous pouvez le sauvegarder depuis le menu `doc`. Pour l'exécuter, revenez à la ligne de commande en tapant la touche esc, tapez par exemple `f(10)`, vous devriez voir s'afficher les carrés de 1 à 10.

Attention dans l'éditeur de programmes, l'appui sur la touche enter interprète le contenu de l'éditeur. Pour passer à la ligne suivante, il faut taper sur la touche de validation (à droite de la touche u).

3ième exemple : Calcul de l'intervalle de confiance de terminale
En syntaxe Xcas. On peut le saisir en ligne de commande

```
F(P,N):=[P-1.96*sqrt(P*(1-P)/N),P+1.96*sqrt
(P*(1-P)/N)]
```

$$(P,N)\mapsto \left[\ P-1.96\sqrt{P\tfrac{1-P}{N}}\quad P+1.96\sqrt{P\tfrac{1-P}{N}}\ \right]$$

On peut éviter les calculs redondants en utilisant une variable locale (utiliser menu Programmes pour saisir fonction, local, return et ffonction)

```
fonction F(P,N)
  local D;
  D:=1.96*sqrt(P*(1-P)/N);
  return [P-D,P+D];
ffonction;
```

Exercice Créez un fichier `carre.py` contenant un script pour afficher un carré avec la tortue. Créez un fichier `carren.py` pour afficher un carré de n pixels, en utilisant une fonction d'argument n et l'instruction `repete`.

Solution Depuis l'éditeur de script, faire `doc` 5 (Effacer). Puis shift-4 `efface`. Ajouter 4 fois `avance; tourne_gauche;`. Appuyer sur enter pour tester. Sauvegardez (`doc` 3 Sauvegarder comme).

Effacer à nouveau (`doc` 5 effacer) Puis shift-4 `efface`. Ajouter avant la ligne `efface`

```
def f(n):
  for j in range(4):
    avance(n)
    tourne_gauche
```

puis après la ligne `efface;` tapez par exemple `f(40)` puis enter. Ensuite faire `doc` 3 (Sauvegardez comme).

Un exemple de fonction non algébrique : le calcul du PGCD de 2 entiers.
Utiliser enter pour passer à la ligne. En syntaxe Xcas

```
fonction pgcd(a,b)
  tantque b!=0 faire
    a,b:=b,irem(a,b);
  ftantque;
  return a;
ffonction
```

On vérifie `pgcd(12345,3425)`

5

. Le même en syntaxe Python

```
def pgcd(a,b):
  while b!=0:
    a,b=b,a
  return a
```

Mise au point
La commande `debug` permet d'exécuter une fonction en mode pas-à-pas, i.e. visualiser l'évolution des variables instruction par instruction, par exemple
`debug(pgcd(12345,3425))`

12.2 Quelques exemples

Le répertoire `Xcas` de l'archive contient quelques exemples de programmes (avec fréquemment une représentation graphique) dont :

- fréquence dans un échantillon, intervalle de fluctuations (freq.xw)
- le paradoxe du duc de Toscane (toscane.xw)
- résolution d'équation du second degré (deg2.xw)
- dichotomie (dicho.xw)
- méthode des rectangles (integr.xw),
- fractale de Mandelbrot (mandel.py)
- un benchmark utilisé par tiplanet pour mesurer la vitesse de l'interpréteur (qcc.xw)

12.3 Commandes utilisables

Contrairement aux adaptations de MicroPython proposées par les constructeurs (dont celui de la TI Nspire), la programmation en (simili-)Python dans KhiCAS n'est pas une application indépendante. Vous pouvez donc utiliser tous les types de Xcas (par exemple les rationnels) et appliquer toutes les commandes de Xcas dans vos programmes. Ceci correspond plus ou moins à un environnement Python avec les modules `math`, `cmath`, `random` (plus complet que le module `urandom` fourni par les constructeurs), `scipy`, `numpy`, un petit module de graphiques pixelisé (`set_pixel(x,y,c)`, `set_pixel()` pour synchroniser l'affichage, `clear()`, `draw_line(x1,y1,x2,y2,c)`, `draw_polygon([[x1,y1],[x2,y2],...],c)`, `draw_rectangle(x,y,w,h,c)`, `draw_circle(x,y,r,c)`, la couleur+epaisseur+remplissage `c` est un paramètre

optionnel, `draw_arc(x,y,rx,ry,t1,t2,c)` permet de tracer un arc d'ellipse). et pour remplacer `matplotlib` on peut utiliser les commande graphiques dans un repère de χCAS (`point`, `line`, `segment`, `circle`, `barplot`, `histogram` et les commandes `plot...`). De plus, vous pouvez travailler avec des expressions et faire du calcul formel dessus. Pour la liste complète des commandes et une présentation détaillée, on renvoie à la documentation de Xcas.

13 Interpréteur MicroPython intégré

χCAS est maintenant fourni avec son propre interpréteur MicroPython (qui n'est pas identique à celui fourni par Numworks). Pour passer dans χCAS de l'interpréteur Xcas à MicroPython et réciproquement vous pouvez taper la commande `python` ou `xcas` dans le shell sur une ligne vide. Ces commandes sont accessibles dans le menu rapide shift).

Attention, si vous lancez MicroPython après avoir travaillé avec Xcas ou/et avec le tableur, il peut ne pas y avoir assez de mémoire pour le tas MicroPython, dans ce cas vous risquez de perdre votre session de travail, pensez à la sauvegarder. Par défaut, 40K sont réservés pour MicroPython. Vous pouvez modifier cette valeur jusqu'à 64K dans la configuration (touche Home puis touche ln). Si vous choisissez une valeur haute, il peut être nécessaire de quitter χCAS et de le réouvrir pour que le tas puisse être alloué dans une zone mémoire contiguë (problème de fragmentation du tas sinon).

Remarque : lorsque l'interpréteur Xcas est actif, si vous passez en argument à la commande `xcas` ou `python` le nom de variable d'une fonction que vous avez programmée, cela affiche son texte source sous forme d'une chaine de caractères en syntaxe Xcas ou Python. Vous pouvez donc programmer en syntaxe Xcas et traduire ensuite en Python si vous devez vous conformer à des règles qui imposent ce langage.

13.1 Les modules standard : math, cmath, random

Ce sont les modules natifs fournis par MicroPython (`urandom` a été renommé `random`), et qui sont conforme au standard. D'autres modules MicroPython standard qui ne sont pas destinés à faire des maths sont disponibles. On peut taper `halp('modules')` pour voir la liste des modules disponibles. Cf l'aide en ligne de MicroPython [8].

13.2 Le module cas

Il donne accès depuis MicroPython aux commandes natives de Xcas. Il contient une seule commande `caseval`, qui prend en argument soit une chaine de caractères qui sera évaluée par l'interpréteur de Xcas, soit une chaine de caractères (contenant le nom de commande à exécuter) puis des arguments de type quelconque (représentant les arguments). Le résultat est une chaine de caractères. On peut bien sur appeler `eval` pour transformer la valeur de retour en objet Python. Exemples :

```
caseval("sin",0)
```

8. `https://docs.micropython.org/`

```
caseval("sin(0)")
caseval("integrate","1/x","x")
caseval("integrate","1/x","x",2,3)
a=[1,2,3]
caseval("a:=",a)
caseval("a")
```

Note : `caseval` possède deux synonymes `xcas` et `eval_expr`.

13.3 Le module graphic

Il s'agit d'un module natif MicroPython qui exporte de Xcas des fonctions de tracé pixelisés. Une partie des commandes est accessible depuis le menu rapide shift `.` Ce module a des synonymes, `kandinsky` et `casioplot` afin de faciliter l'utilisation de scripts Python pour calculatrices Numworks (Epsilon) et Casio.

13.4 Le module matplotl

Il s'agit d'un module natif MicroPython qui exporte de Xcas des fonctions de tracé repéré et vise à une certaine compatibilité avec le module `matplotlib` (ou un de ses sous-modules) sur PC. Une partie des commandes est accessible depuis le menu rapide shift 0.

13.5 Le module arit

C'est un module natif qui exporte de Xcas des fonctions d'arithmétique entière : test de primalité et prochain nombre premier (par Miller-Rabin), factorisation d'entiers pas trop gros (détection par Pollard-rho du plus petit facteur premier, donc jusqu'à environ 9 chiffres), pgcd et identité de Bèzout, indicatrice d'Euler (si on sait factoriser). J'y ai aussi inclus deux fonctions de conversion liste vers chaine de caractères pour faciliter l'enseignement d'un peu de cryptographie.

13.6 Le module linalg

Il permet de manipuler les listes comme des vecteurs et les listes de listes comme des matrices. Contrairement à Xcas, Python n'est pas un langage spécifiquement adapté aux maths, il faut utiliser des commandes préfixées `add, sub, mul` pour effectuer les opérations arithmétiques de base `+ - *` sur les vecteurs et matrices représentés par des listes.

Le module `linalg` est un module natif, qui utilise Xcas pour effectuer la quasi-totalité des calculs.

13.7 Le module numpy

C'est une surcouche du module `linalg` qui définit une classe `array` pour représenter les vecteurs et les matrices. On peut alors utiliser `+ - *` pour faire les opérations de base sur vecteurs et matrices.

Le module `numpy` n'est pas un module natif, c'est un texte source écrit en Python. Son importation consomme donc de la mémoire RAM. Vous pouvez écrire votre propre version de `numpy.py` et la stocker dans le scriptstore, elle prendra alors la précédence sur la version utilisée par défaut. Cette dernière vise à assurer un minimum de compatibilité avec le module du même nom sur PC. Bien que non natif, ce module est disponible en mode examen (le texte source par défaut est intégré au code source de l'interpréteur MicroPython).

```
import linalg
import math
class array:
    def __init__(self, a):
        self.a = a

    def __add__(self, other):
        return array(linalg.add(self.a , other.a))

    def __sub__(self, other):
        return array(linalg.sub(self.a , other.a))

    def __mul__(self, other):
        if type(self)==array:
            if type(other)==array:
                return array(linalg.mul(self.a , other.a))
            return array(linalg.mul(self.a,other))
        return array(linalg.mul(self,other.a))

    def __rmul__(self, other):
        if type(self)==array:
            if type(other)==array:
                return array(linalg.mul(self.a , other.a))
            return array(linalg.mul(self.a,other))
        return array(linalg.mul(self,other.a))

    def __matmul__(self, other):
        return __mul(self,other)

    def __getitem__(self,key):
        r=(self.a)[key]
        if type(r)==list or type(r)==tuple:
            return array(r)
        return r

    def __setitem__(self, key, value):
        if (type(value)==array):
            (self.a)[key]=value.a
```

```
        else:
            (self.a)[key]=value
        return None

    def __len__(self):
        return len(self.a)

    def __str__(self):
        return 'array('+str(self.a)+')'

    def __repr__(self):
        return 'array('+str(self.a)+')'

    def __neg__(self):
        return array(-self.a)

    def __pos__(self):
        return self

    def __abs__(self):
        return array(linalg.abs(self.a))

    def __round__(self):
        return array(linalg.apply(round,self.a,linalg.matrix))

    def __trunc__(self):
        return array(linalg.apply(trunc,self.a,linalg.matrix))

    def __floor__(self):
        return array(linalg.apply(floor,self.a,linalg.matrix))

    def __ceil__(self):
        return array(linalg.apply(ceil,self.a,linalg.matrix))

    def T(self):
        return array(linalg.transpose(self.a))

def real(x):
    if type(x)==array:
        return array(linalg.re(x.a))
    return x.real

def imag(x):
    if type(x)==array:
        return array(linalg.im(x.a))
    return x.imag
```

```
def conj(x):
    if type(x)==array:
        return array(linalg.conj(x.a))
    return linalg.conj(x)

def sin(x):
    if type(x)==array:
        return array(linalg.apply(math.sin,x.a,linalg.matrix))
    return math.sin(x)

def cos(x):
    if type(x)==array:
        return array(linalg.apply(math.cos,x.a,linalg.matrix))
    return math.cos(x)

def tan(x):
    if type(x)==array:
        return array(linalg.apply(math.tan,x.a,linalg.matrix))
    return math.tan(x)

def asin(x):
    if type(x)==array:
        return array(linalg.apply(math.asin,x.a,linalg.matrix))
    return math.asin(x)

def acos(x):
    if type(x)==array:
        return array(linalg.apply(math.acos,x.a,linalg.matrix))
    return math.acos(x)

def atan(x):
    if type(x)==array:
        return array(linalg.apply(math.atan,x.a,linalg.matrix))
    return math.atan(x)

def sinh(x):
    if type(x)==array:
        return array(linalg.apply(math.sinh,x.a,linalg.matrix))
    return math.sinh(x)

def cosh(x):
    if type(x)==array:
        return array(linalg.apply(math.cosh,x.a,linalg.matrix))
    return math.cosh(x)
```

```
def tanh(x):
    if type(x)==array:
        return array(linalg.apply(math.tanh,x.a,linalg.matrix))
    return math.tanh(x)

def exp(x):
    if type(x)==array:
        return array(linalg.apply(math.exp,x.a,linalg.matrix))
    return math.exp(x)

def log(x):
    if type(x)==array:
        return array(linalg.apply(math.log,x.a,linalg.matrix))
    return math.log(x)

def size(x):
    if type(x)==array:
        return linalg.size(x.a)
    return linalg.size(x)

def shape(x):
    if type(x)==array:
        return linalg.shape(x.a)

def dot(a,b):
    return a*b

def transpose(a):
    if type(x)==array:
        return array(linalg.transpose(x.a))

def trn(a):
    if type(x)==array:
        return array(linalg.conj(linalg.transpose(x.a)))
    return linalg.conj(linalg.transpose(x.a))

def zeros(n,m=0):
    return array(linalg.zeros(n,m))

def ones(n,m=0):
    return array(linalg.ones(n,m))

def eye(n):
    return array(linalg.eye(n))

def det(x):
```

```
    if type(x)==array:
        return linalg.det(x.a)
    return linalg.det(x)

def inv(x):
    if type(x)==array:
        return array(linalg.inv(x.a))
    return linalg.inv(x)

def solve(a,b):
    if type(a)==array:
        if type(b)==array:
            return array(linalg.solve(a.a,b.a))
        return array(linalg.solve(a.a,b))
    if type(b)==array:
        return array(linalg.solve(a,b.a))
    return linalg.solve(a,b)

def eig(a):
    if type(a)==array:
        r=linalg.eig(a.a)
        return array(r[0]),array(r[1])
    return linalg.eig(a)

def linspace(a,b,c):
    return array(linalg.linspace(a,b,c))

def arange(a,b,c=1):
    return array(linalg.arange(a,b,c))

def reshape(a,n,m=0):
    if type(n)==tuple:
        m=n[1]
        n=n[0]
    if type(a)==array:
        return array(linalg.matrix(n,m,a.a))
    return linalg.matrix(n,m,a)
```

14 Applications additionnelles.

Accessibles par le menu `doc` 1 ou le raccourci clavier shift-ANS. Il y a actuellement la géométrie, le tableur, le tableau périodique des éléments (d'après Maxime Friess), et des exemples d'addin (dont le code source peut servir de modèle aux programmeurs expérimentés souhaitant programmer des addins pour χCAS) : une application finance, la suite de Syracuse (très simple), la fractale de Mandelbrot, un jeu de

mastermind (qui peut aussi servir de patience),

15 Raccourcis claviers.

— la touche de validation à droite de la touche u n'a pas toujours la même signification que la touche enter, en particulier dans l'éditeur de programmes elle permet de passer à la ligne et dans l'éditeur d'expressions elle évalue la sélection
— la touche `trig` et ses shift/ctrl donnent accès aux fonctions `sin, cos, tan`
— on peut taper ctrl-`*` pour saisir `"` et shift-`*` pour saisir `'`
— le caractère `\` est accessible via ctrl-`/`, `%` par shift-`/`.
— `;` et `:` sont en shift/ctrl de `,`
— shift-1 à shift-6 : selon le mode (Xcas ou Python) voir les légendes
— mode Xcas : shift-7 matrices, 8 complexes, 9 arithmétique, `0` proba et autres, `.` réels, `(-)` polynomes, `(` listes, `)` programmation
— mode Python : shift-7 et 8 matrices, 9 arithmétique, 0 graphes repérés, `.` graphes pixelisés, `(-)` couleurs, `(` listes, `)` programmation
— changement d'interpréteur : shift `)` 8
— `=>` suivi d'une unité physique effectue une conversion d'unité, `=>` suivi d'une fonction permet d'exécuter des actions :
`=>*` écriture sous forme de produit (pour une unité, écriture en utilisant les unités fondementales du système MKSA),
`=>+` écriture sous forme de somme (pour une unité, écriture sous forme le plus simple possible),
`=>/` écriture sous forme de quotient,
`=>sin`, `=>cos`, `=>tan` conversion vers des sinus, cosinus ou tangentes.
`=>,` permet de chronométrer une évaluation,

— `16=>=>` écrit les entiers qui suivent en base 16, `10=>=>` en base 10, `8=>=>` en base 8
— ctrl puis `p` programmation,
— ctrl puis o donne accès au menu d'options
— ctrl suivi de R dans le shell permet de ré-exécuter la session actuelle (run)
— ctrl suivi de S dans le shell permet d'accéder à la configuration (setup)
— ctrl suivi de Z permet en général d'annuler la dernière modification

Dans l'éditeur de programmes :

— touches curseur shiftées : déplacement en début/fin de ligne/fichier.
— ctrl suivi de C permet de débuter une sélection, refaire ctrl puis C pour copier, ctrl puis X ou del pour couper, ctrl puis V pour coller
— enter : si une recherche/remplacement de mot est active (après avoir fait menu 6), recherche l'occurence suivante d'un mot. Sinon, passe à la ligne.
— DEL efface le caractère précédent ou la sélection.
— ans (shift (-)) : bascule entre l'éditeur et la figure tortue
— esc : quitte l'éditeur et revient en ligne de commandes. On peut revenir ensuite à l'éditeur en tapant à nouveau esc.

16 Extinction, reset, horloge.

Comme tout logiciel, χCAS n'est pas exempt de bugs. Si un calcul bloque, commencez par essayer de l'interrompre en appuyant sur la touche ON. Si cela ne suffit pas, il faut se résoudre à appuyer sur le bouton RESET à l'arrière de la calculatrice. Pour pouvoir relancer χCAS, il faut au préalable réactiver ndless (depuis `2. Mes documents` puis dans le répertoire `ndless`).

Pour remettre à l'heure l'horloge interne de la calculatrice qui est affichée dans le shell, connectez votre calculatrice avec votre ordinateur ou tapez une commande du type `hh,mm=>`, par exemple `16,05=>`, pour 16h05.

Sur les modèles CX uniquement, le rétro-éclairage de la calculatrice s'éteint automatiquement au bout de quelques dizaines de secondes si on ne fait rien, il suffit de taper sur ON pour rallumer. On peut aussi faire ctrl ON pour forcer l'extinction du rétro-éclairage. Attention, il ne s'agit pas d'une vraie extinction de la calculatrice (il n'y a pas assez d'informations disponibles sur le matériel de Texas Instruments pour le faire). Si vous voulez vraiment éteindre la calculatrice, il faut quitter χCAS en tapant autant de fois que nécessaire la touche doc (entre 2 et 4 fois). Pour éviter une décharge trop rapide de la batterie, sur les nspire CX, χCAS se quitte de lui-même au bout de 2 heures d'inactivité (en sauvegardant la session courante sous le nom habituel `session.xw.tns`). La calculatrice se rallume alors brièvement avant de s'éteindre réellement.

En cas d'inactivité prolongée de plusieurs jours, la calculatrice reboote quand on la rallume, il faut alors reactiver ndless avant de pouvoir utiliser χCAS.

17 Copyright, licences et remerciements

— Giac et χCAS, noyau de calcul (c) B. Parisse et R. De Graeve, 2019. Les calculs en entiers sont effectués avec GMP [9], les flottants multipécision avec MPFR [10], l'arithmétique d'intervalle avec MPFI [11]

— Interface de χCAS adaptée par B. Parisse à partir de l'interface utilisateur du code source d'Eigenmath créée par Gabriel Maia et de l'interface utilisateur de Xcas.
License d'utilisation de χCAS : GPL2. (voir le détail des conditions dans le fichier LICENSE.GPL2 [12] ou sur la page GPL2 [13] du site de la Free Software Foundation).

— Remerciements à Fabian Vogt pour firebird-emu, ndless. Remerciements à toute l'équipe de ndless. Remerciement à l'équipe de tiplanet, en particulier Xavier Andréani, Lionel Debroux et Adrien Bertrand, pour le forum de discussion et tout le travail de mise en valeur de χcas (stockages d'archives, articles, concours).

9. `https://gmplib.org/`
10. `https://www.mpfr.org/`
11. `https://gforge.inria.fr/projects/mpfi/`
12. `https://www-fourier.ujf-grenoble.fr/~parisse/LICENSE.GPL2`
13. `https://www.gnu.org/licenses/old-licenses/gpl-2.0.fr.html`

— le tableau périodique des éléments est un portage de l'application de Maxime Friess [14] avec son autorisation de diffusion sous licence GPL.

18 Développement en C++ avec χCAS et Ndless

Il faut tout d'abord installer le SDK de ndless [15], sous linux par la commande
`git clone --recursive https://github.com/ndless-nspire/Ndless.git`
Puis il faut le compiler, ce qui prend une à plusieurs heures selon la puissance de l'ordinateur :

```
cd Ndless/ndless-sdk/toolchain/
./build_toolchain.sh
```

Ensuite il faut installer le code source de Giac/Xcas [16].

```
wget https://www-fourier.univ-grenoble-alpes.fr/~parisse/giac/giac_stable.tgz
tar xvfa giac_stable.tgz
cd giac-1.9.0/micropython-1.12/nspire
```

ouvrir le fichier `mklib` et ajuster le chemin pour recopier la librairie `libmicropy.a`,
puis

```
sh mklib
cd ../../src
cp config.h.nspire config.h
cp Makefile.nspire Makefile
```

ajustez les chemins dans le fichier `Makefile` puis tapez `make`.

À compléter.

14. https://github.com/M4xi1m3/nw-atom
15. https://hackspire.org/index.php/C_and_assembly_development_introduction
16. https://www-fourier.univ-grenoble-alpes.fr/~parisse/giac_fr.html