Chapitre 5

Équations

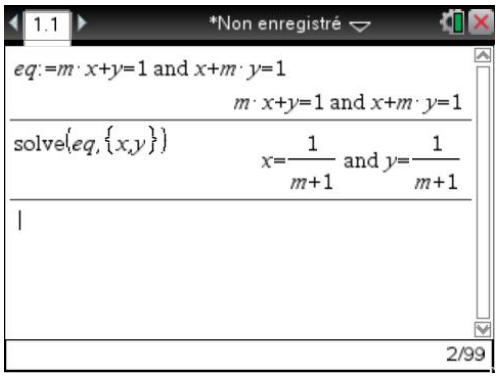
On dispose d'un choix important de fonctions de résolution d'équations ou de systèmes dans $\mathbb{R}$ ou dans $\mathbb{C}$. Les fonctions **solve** et **cSolve** s'utilisent avec la même syntaxe et permettent d'obtenir les solutions sous forme d'égalités, alors que **zeros** et **cZeros** donnent les résultats sous forme d'une liste, ou d'une matrice dans le cas de systèmes d'équations. Vous trouverez un résumé des syntaxes à utiliser dans le **chapitre 16** de ce document, ainsi que de nombreux exemples d'utilisation dans les différents chapitres. Nous allons étudier ici deux types d'équations un peu délicates.

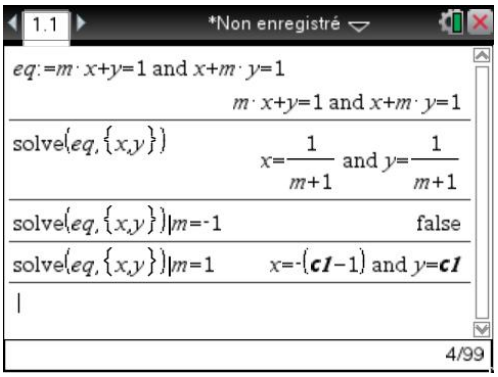
1. Systèmes

1.1 *Systèmes linéaires avec paramètres*

Il faut être particulièrement prudent lors de la résolution de systèmes linéaires utilisant des paramètres. Les calculs effectués par l'unité nomade TI-Nspire CAS peuvent très bien ne pas être valables pour certaines valeurs du paramètre. Ce problème a déjà été abordé dans le **chapitre 2**, dans le paragraphe "Les risques de la simplification automatique".

Nous avons vu que pour le système d'équations $\begin{cases} m\,x + y = 1 \\ x + m\,y = 1 \end{cases}$ la solution obtenue n'est pas toujours valide.[1]





[1] Lors de la saisie des équations ne pas omettre le signe multiplié entre *m* et la variable *x* par exemple, sinon ce serait considéré comme une variable *mx*.

Dans le premier écran, il semble qu'il y ait une seule solution : $x=\dfrac{1}{m+1}$ et $y=\dfrac{1}{m+1}$.

Le second écran montre que la situation est différente pour $m=1$ ou pour $m=-1$.

- Pour $m=-1$, la réponse obtenue, **false**, indique que le système n'a pas de solution.
- Pour $m=1$, il y a au contraire une infinité de solutions, du type $x=-(t-1)$, $y=t$, avec t quelconque.

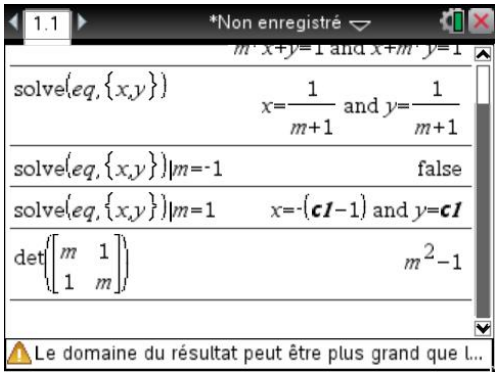
La TI-Nspire CAS utilise une notation du type ***c1***, ***c2***, … pour désigner un nombre quelconque. À chaque fois qu'une telle variable doit être utilisée pour exprimer les solutions d'une équation, le compteur utilisé pour les numéroter augmente de 1. Après ***c255***, on revient à ***c1***.

D'après la forme finale de la solution, seul le fait que $m=-1$ soit un cas particulier était prévisible. En effet, l'expression générale des solutions n'est pas définie lorsque le dénominateur $m+1$ est nul. En revanche, rien ne laisse prévoir le cas particulier $m=1$.

Pour mettre en évidence les cas particuliers dans un système linéaire où intervient un paramètre, il faut par exemple calculer le déterminant de ce système.

Pour le faire, il suffit d'utiliser la fonction **det**. Voici deux exemples d'utilisation.

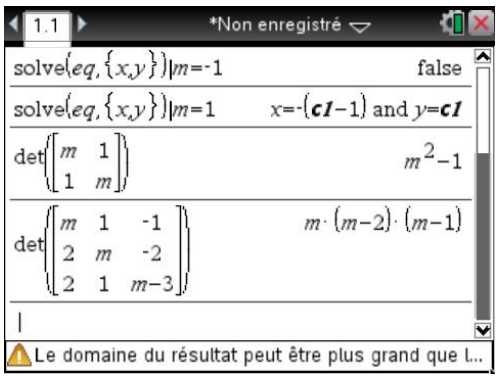
1. Pour le système d'équations $\begin{cases} mx+y=1 \\ x+my=1 \end{cases}$, on entrera **`det([m,1;1,m])`** (touche ?! pour le ;)



2. Pour le système d'équations $\begin{cases} mx+y-z=m \\ 2x+my-2z=2 \\ 2x+y+(m-3)z=2 \end{cases}$, on entrera

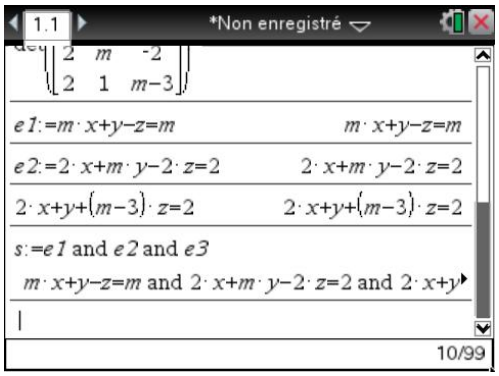
   `det([m,1,-1;2,m,-2;2,1,m-3])`

   (on peut aussi utiliser le modèle permettant la saisie des matrices, en appuyant sur)

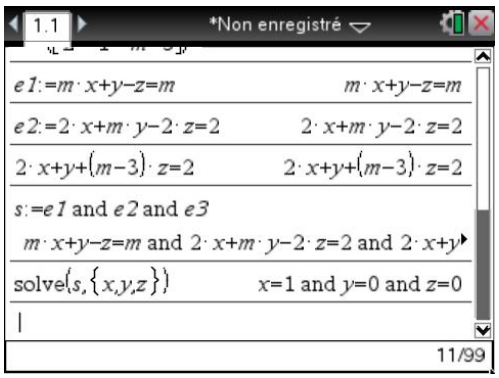


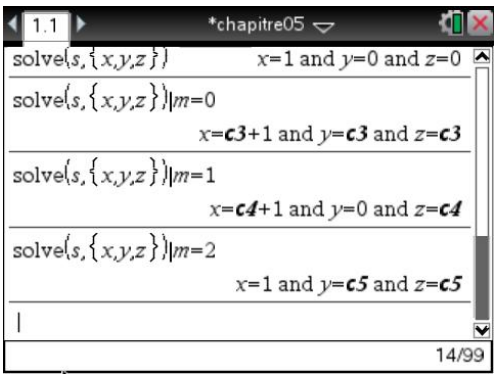
À partir de là, il est facile de prévoir les cas à traiter à part, pour chaque système.

Voici par exemple la résolution du second système, obtenue en utilisant la fonction **solve**.



☞ *On a choisi ici de définir d'abord les équations du système, puis le système lui-même à partir des équations qui le composent. Ce n'est pas indispensable, mais peut parfois être plus pratique.*
N.B. *On peut utiliser* **and** *comme cela a été fait ici, ou le modèle permettant de saisir un système d'équations, comme nous le ferons dans la suite.*





☞ *Lors des trois dernières résolutions, on a obtenu une expression en fonction de* ***c2****,* ***c3*** *et de* ***c4****.*

On obtient une solution unique, $S=\{(1,0,0)\}$, dans "le cas général", c'est-à-dire pour m distinct de 0, 1 et 2.

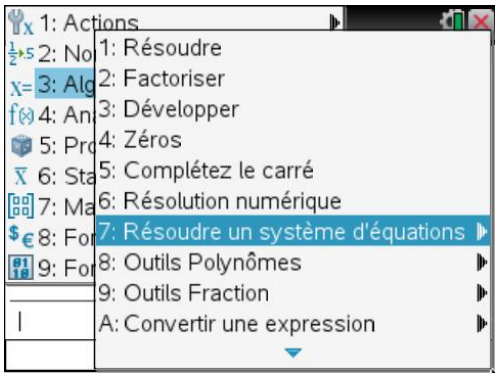
En revanche,

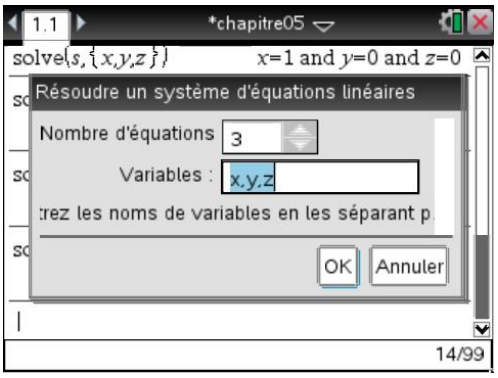
- pour $m=0$, $S=\{(t+1,t,t),\ t\in\mathbb{R}\}$
- pour $m=1$, $S=\{(t+1,0,t),\ t\in\mathbb{R}\}$
- pour $m=2$, $S=\{(1,t,t),\ t\in\mathbb{R}\}$

La résolution "sans précaution" de ce système conduirait donc à un résultat faux dans ces trois cas particuliers.

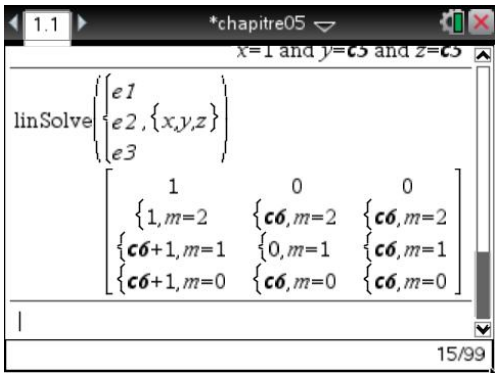
Pour résoudre un système linéaire, pour le saisir on peut procéder de la façon suivante :

menu 3 7 2





Cette procédure fait appel à la fonction **linSolve**, le résultat est donné sous forme de matrice :



1.2 *Systèmes non linéaires*

La TI-Nspire CAS utilise une méthode sophistiquée de résolution des systèmes non linéaires. Elle permet de résoudre certains systèmes dont les équations sont des fonctions polynomiales des inconnues.

Voici par exemple la résolution de $\begin{cases} x \cdot y = 7 \\ x^2 - y^2 = -2 \end{cases}$

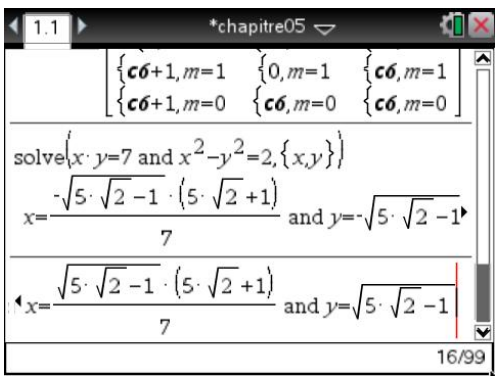
On peut l'obtenir en utilisant la fonction **solve** ou la fonction **zeros**.

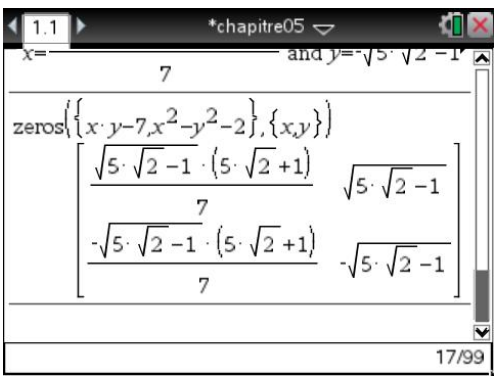
Dans le premier cas, voir écran de gauche (une copie du résultat permet de voir la fin de la réponse), on écrit :

```
solve(x*y=7 and x^2-y^2=-2,{x,y})
```

dans le second, voir écran de droite, on écrit :

```
zeros({x*y-7,x^2-y^2+2},{x,y})
```





Il y a un couple de solutions : $S=\{(x_1,y_1),(x_2,y_2)\}$.

Avec la fonction **solve**, on obtient un résultat sous forme d'égalités :

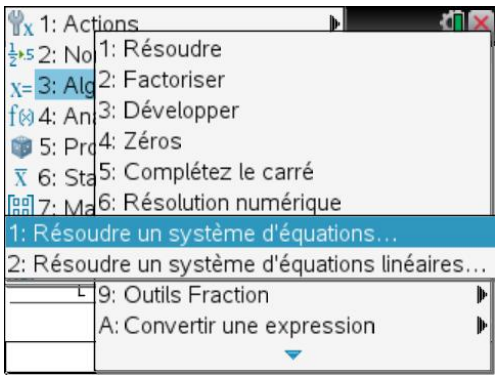
$$x=x_1 \text{ and } y=y_1 \text{ or } x=x_2 \text{ and } y=y_2$$

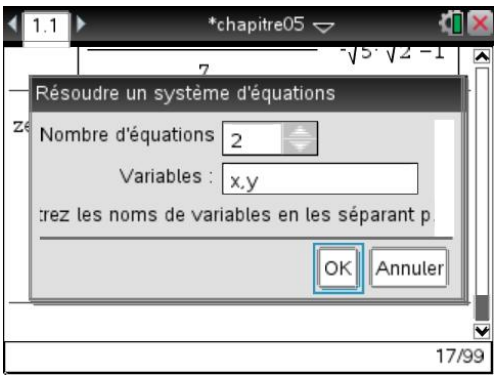
Avec la fonction **zeros**, les valeurs sont placées dans une matrice du type :

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

Pour la saisie d'un système non linéaire, on peut aussi procéder de la façon suivante :

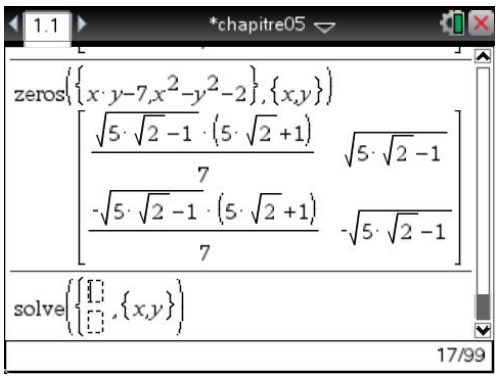
menu 3 7 1

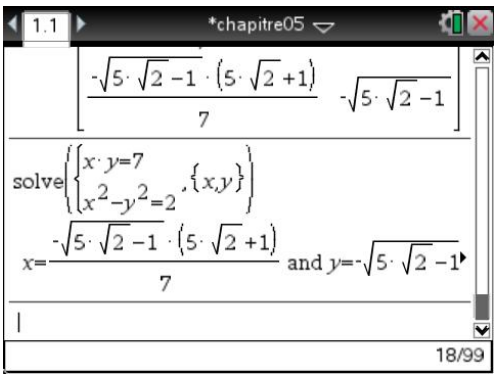




La procédure fait appel à la fonction **solve**.

Il ne reste plus qu'à saisir les deux équations dans les emplacements prévus dans le modèle et à valider.

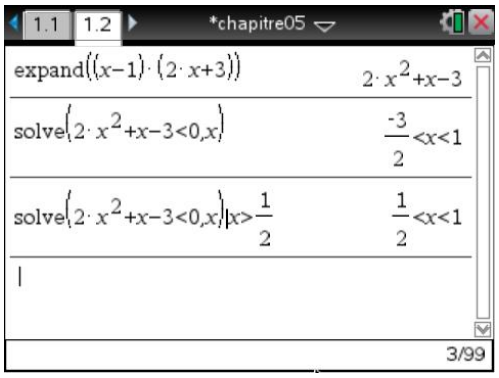


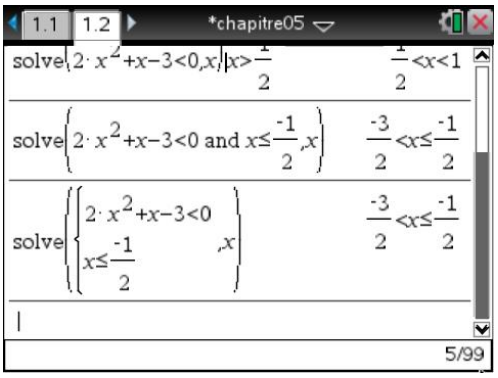


2. Inéquations

La fonction **solve** permet également de résoudre des inéquations. Voici un exemple simple sur l'écran de gauche et sur l'écran de droite la résolution d'un système en utilisant la syntaxe *inéq1* and *inéq2*, et dans un deuxième temps l'utilisation du modèle ( ).

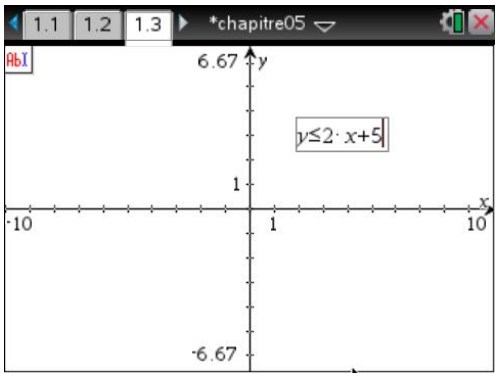
Les symboles servant à écrire les inéquations se trouvent dans la palette accessible à l'aide de la combinaison de touches ctrl (seconde fonction de la touche =).

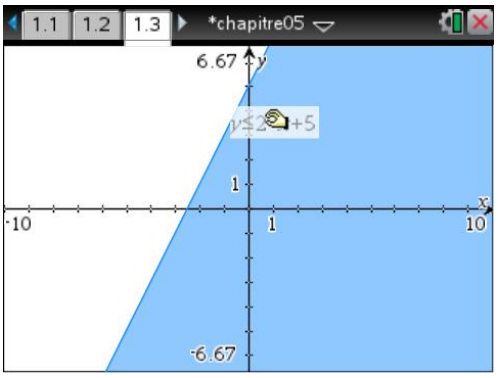




L'application Graphiques permet de représenter des inéquations.

On peut par exemple saisir le texte d'une inéquation et rapprocher le texte des axes, afin d'obtenir la représentation de la partie du plan vérifiant l'inégalité.

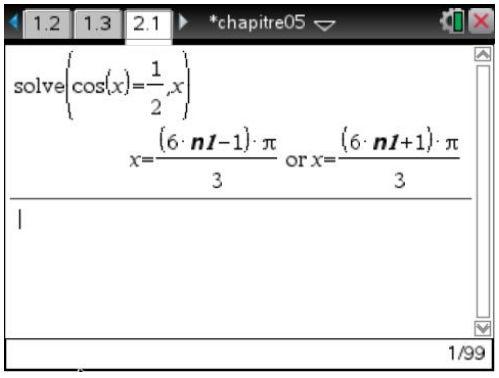




3. Équations trigonométriques

3.1 *Résolution symbolique*

Voici par exemple les racines de l'équation $\cos x = \frac{1}{2}$:

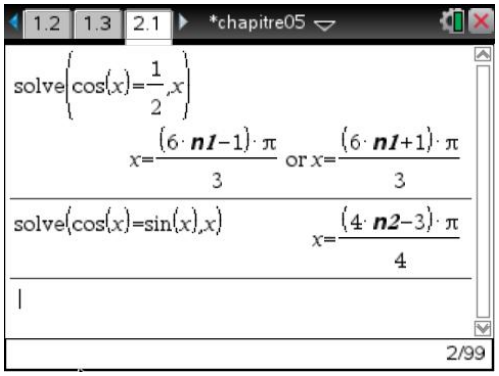


☞ *Bien vérifier que le mode angulaire est réglé sur Radian.*

L'expression des solutions fait intervenir la variable **n1** qui désigne un entier quelconque.

Ici, on a donc obtenu $x = \frac{6n+1}{3}\pi$ ou $x = \frac{6n-1}{3}\pi$ avec *n* entier quelconque.

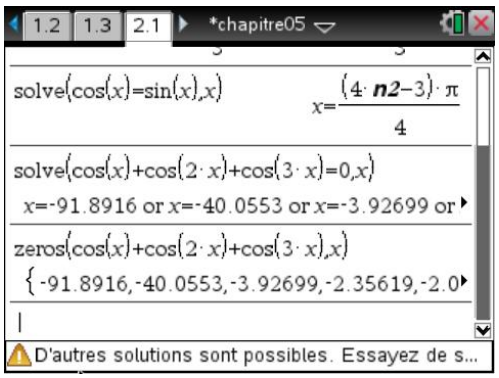
Lors de la résolution suivante, cette variable sera désignée par **n2**, et ainsi de suite jusqu'à **n255**. Le compteur utilisé par la TI-Nspire CAS pour numéroter ces variables entières est alors remis à 1.



3.2 *Résolution approchée*

Lorsque la TI-Nspire CAS ne sait pas résoudre l'équation proposée, elle passe en mode de résolution approchée.

Essayons de résoudre $\cos(x)+\cos(2x)+\cos(3x)=0$:



Comme on peut le voir, on obtient seulement la valeur numérique approchée de certaines solutions. Lorsque l'on vient juste d'effectuer le calcul, le message **D'autres solutions sont possibles…** est affiché en bas de l'écran. Il signifie que d'autres solutions pourraient exister, et que l'on ne peut donc pas vraiment se fier au résultat affiché !

3.3 *Résolution symbolique assistée par la TI-Nspire CAS*

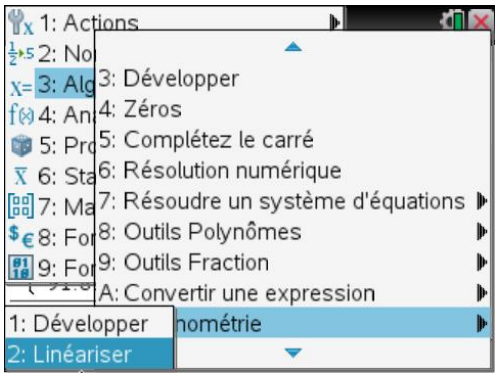
Dans le dernier exemple, la TI-Nspire CAS n'a pas résolu cette équation trigonométrique. Pour la résoudre, on peut factoriser l'expression.

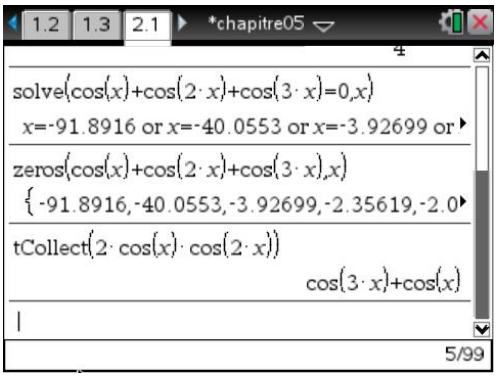
On a l'égalité

$$\cos(x)+\cos(3x)=2\cos\left(\frac{3x+x}{2}\right)\cos\left(\frac{3x-x}{2}\right)=2\cos(x)\cos(2x)$$

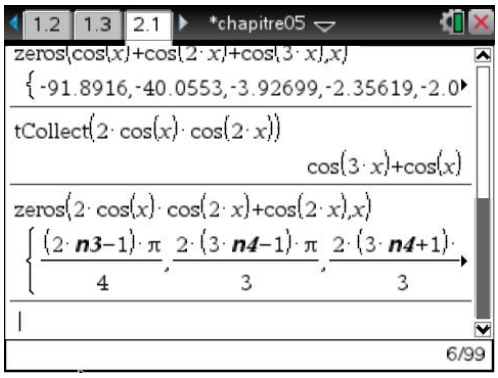
Il est possible de le vérifier :

menu 3 B 2

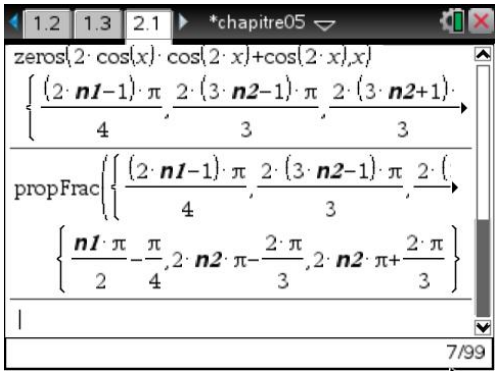




On peut donc écrire cette équation sous la forme $\cos(2x)+2\cos(x)\cos(2x)=0$ et la résolution est alors immédiate :



Il est possible d'obtenir une forme plus classique des solutions en appliquant la fonction **propFrac**, présente dans le menu **Algèbre** (menu 3 9 1), à l'expression précédente.



En conclusion, les solutions sont les nombres du type
$x=-\frac{2\pi}{3}+2k\pi$, $x=\frac{2\pi}{3}+2k\pi$ ou $x=-\frac{\pi}{4}+k\frac{\pi}{2}$, avec k entier relatif arbitraire.

Par exemple, dans $[-\pi,\pi]$, on trouve : $-\frac{2\pi}{3}$, $\frac{2\pi}{3}$, $-\frac{\pi}{4}-\frac{\pi}{2}=-\frac{3\pi}{4}$, $-\frac{\pi}{4}$, $-\frac{\pi}{4}+\frac{\pi}{2}=\frac{\pi}{4}$ et $-\frac{\pi}{4}+2\frac{\pi}{2}=\frac{3\pi}{4}$.

Dans le calcul précédent, nous avons utilisé l'égalité :

$$\cos(x)+\cos(3x)=2\cos\left(\frac{3x+x}{2}\right)\cos\left(\frac{3x-x}{2}\right)=2\cos(x)\cos(2x)$$

Il est certes possible de le *vérifier* avec la TI-Nspire CAS, mais c'est nous qui avons dû avoir l'idée d'utiliser l'égalité

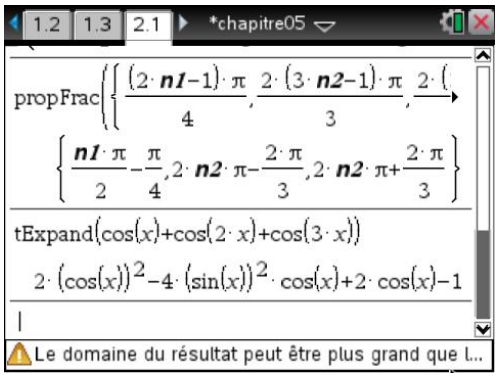
$$\cos(p)+\cos(q)=2\cos\left(\frac{p+q}{2}\right)\cos\left(\frac{p-q}{2}\right)$$

et surtout penser à l'utiliser astucieusement en regroupant le premier et le troisième terme de la somme $\cos(x)+\cos(2x)+\cos(3x)$.

Nous allons maintenant voir deux autres méthodes qui seront généralement efficaces dans les équations que vous pourriez avoir à résoudre, même si elles sont parfois assez différentes de celles que l'on utiliserait à la main.

3.4 *Utilisation de la fonction tExpand*

On commence par se ramener à une équation polynomiale en $\sin(x)$ et $\cos(x)$ en développant l'expression à l'aide de la fonction **tExpand** (menu 3 B 1).



On a obtenu :

$$\cos(x)+\cos(2x)+\cos(3x)=2\cos^2(x)-4\sin^2(x)\cos(x)+2\cos(x)-1$$

On a pour l'instant un "cocktail" de sinus et de cosinus. Il serait plus simple d'avoir seulement l'un des deux… Nous allons tout exprimer en fonction de $t=\cos(x)$ en utilisant l'égalité entre $\sin^2(x)$ et $1-\cos^2(x)$. On peut également utiliser la fonction de conversion (▸**cos**) pour tout écrire en fonction de cosinus.

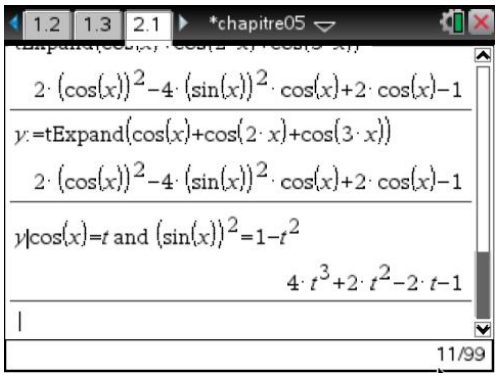
Reprenons, pour plus de clarté, le calcul depuis le début :

Pour commencer, on mémorise dans la variable y l'expression obtenue en appliquant **tExpand** à $\cos(x)+\cos(2x)+\cos(3x)$:

```
y:=tExpand(cos(x)+ cos(2x)+ cos(3x))
```

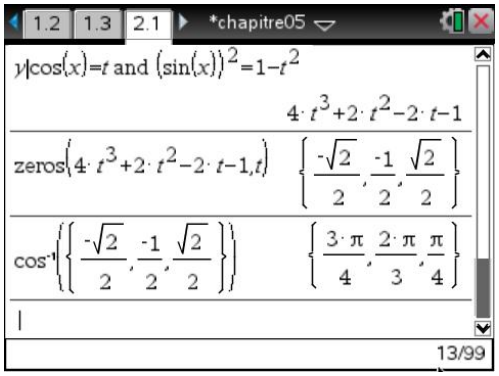
Ensuite, on demande d'exprimer y en fonction de $t=\cos(x)$ en utilisant l'instruction :

```
y|cos(x)=t and sin(x)^2=1-t^2
```



Il reste à rechercher les racines de cette expression, puis à déterminer les valeurs de *x* correspondant à ces racines.

On utilise pour cela le fait que $\cos(x)=t$ est équivalent à $x=\arccos(t)+2n\pi$ ou $x=-\arccos(t)+2n\pi$.



On obtient ainsi les solutions

$$x=\frac{2\pi}{3}+2n\pi,\ x=-\frac{2\pi}{3}+2n\pi,\ x=\frac{3\pi}{4}+2n\pi$$
$$x=-\frac{3\pi}{4}+2n\pi,\ x=\frac{\pi}{4}+2n\pi,\ x=-\frac{\pi}{4}+2n\pi$$

avec *n* entier quelconque.

De plus $x=\frac{3\pi}{4}+2n\pi=-\frac{\pi}{4}+\pi+2n\pi=-\frac{\pi}{4}+(2n+1)\pi$, les solutions obtenues à partir de $x=-\frac{\pi}{4}+2n\pi$ et de $x=\frac{3\pi}{4}+2n\pi$ se regroupent donc sous la forme $x=-\frac{\pi}{4}+n\pi$.

Il en est de même pour $x=-\frac{3\pi}{4}+2n\pi$ et $x=\frac{\pi}{4}+2n\pi$.

En conclusion, il reste :

$$x=\frac{2\pi}{3}+2n\pi,\ x=-\frac{2\pi}{3}+2n\pi,\ x=-\frac{\pi}{4}+n\pi,\ x=\frac{\pi}{4}+n\pi$$

Et il est possible de regrouper les deux dernières sous la forme $x=-\frac{\pi}{4}+n\frac{\pi}{2}$. On retrouve ainsi les solutions obtenues lors de la précédente résolution.

3.5 *Utilisation de tan(x/2)*

Une variante de cette méthode consiste à utiliser les relations

$$\cos(x)=\frac{1-t^2}{1+t^2}\quad \sin(x)=\frac{2t}{1+t^2}\quad \text{avec } t=\tan\left(\frac{x}{2}\right)$$

Ces formules sont utilisables sous réserve que *x* ne soit pas de la forme $(2k+1)\pi$, avec *k* entier relatif.

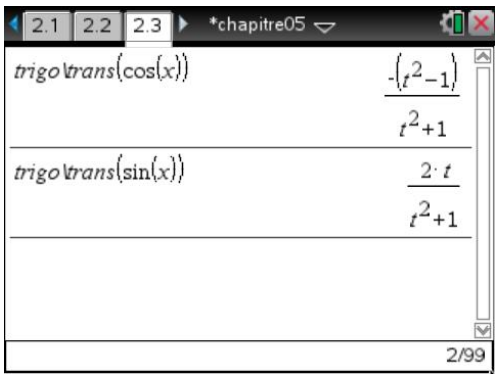
Cette méthode est assez générale mais conduit souvent à des équations de degré plus élevé.

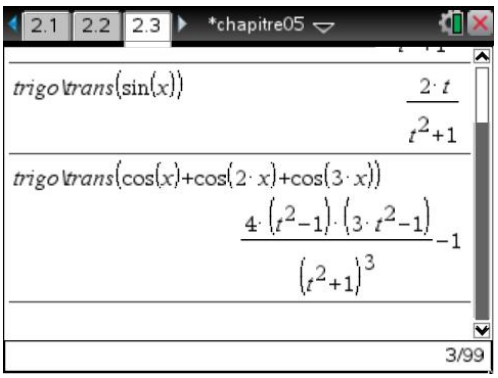
On peut écrire une courte fonction permettant d'automatiser cette transformation :

```
Define LibPub trans(ex)=Func
©ex : cos x, sin x → f(tan(x/2))
Local y
y:=tExpand(ex)
y|cos(x)=(1-t^2)/(1+t^2) and sin(x)=2*t/(1+t^2)
EndFunc
```

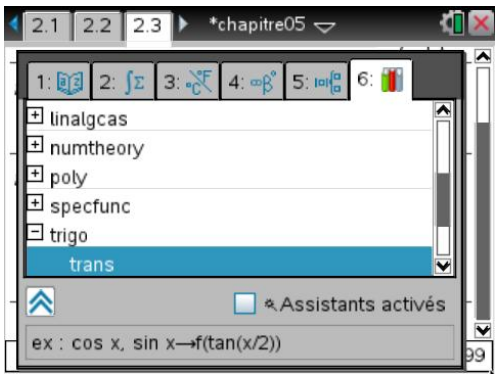
☞ *Voir* ***chapitre 14*** *pour plus d'information sur la programmation, et* ***chapitre 15*** *pour l'utilisation de l'instruction* **Define LibPub** *permettant de faire apparaître la fonction dans le catalogue.*

Voici quelques exemples d'utilisation de cette fonction :





☞ *Le commentaire placé en début de programme, juste après* **Func** *ou* **Prgm** *est affiché en bas de l'écran lorsque l'on demande la liste de fonctions définies par l'utilisateur (catalogue :* 📖 **6** *).*



Nous avons généralement utilisé des commentaires du type *arg*1,*arg2*,... : *description*

3.6 *Une fonction de résolution*

Il est possible d'écrire une fonction de résolution permettant de déterminer les racines de certaines équations trigonométriques en utilisant la méthode précédente.

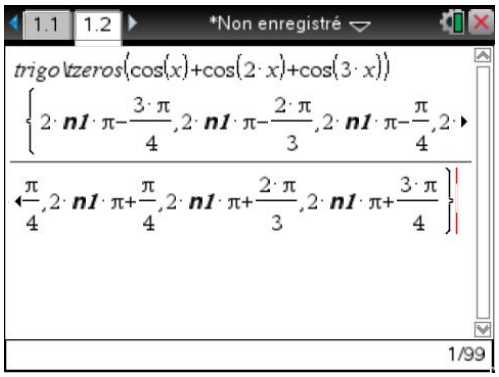
```
Define LibPub tzeros(ex)=Func
©ex : zeros d'une expr. trig.
Local y
y:=tExpand(ex)
```

```
y:=y|cos(x)=(1−t^2)/(1+t^2) and sin(x)=2*t/(1+t^2)
2*tan⁻¹(zeros(y,t))+2*n1*π
EndFunc
```

☞ *Voir* ***chapitre 14*** *pour plus d'information sur la programmation, et* ***chapitre 15*** *pour l'utilisation de l'instruction* **Define LibPub** *permettant de faire apparaître la fonction dans le catalogue.*

Voici par exemple la résolution de l'équation $\cos(x)+\cos(2x)+\cos(3x)=0$:

tzeros(cos(x)+cos(2x)+cos(3x))

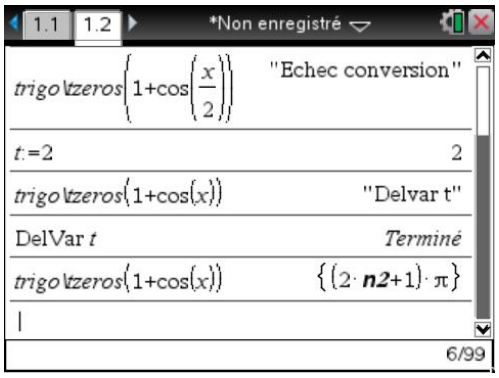


On retrouve bien les valeurs obtenues au paragraphe précédent : $x=\frac{\pi}{4}+2n\pi$, $x=\frac{3\pi}{4}+2n\pi$, $x=-\frac{\pi}{4}+2n\pi$, $x=-\frac{3\pi}{4}+2n\pi$, $x=\frac{2\pi}{3}+2n\pi$, $x=-\frac{2\pi}{3}+2n\pi$.

Plusieurs problèmes peuvent se présenter lors de l'utilisation de cette première version :

- Pour que cette fonction soit utilisable, il faut impérativement que les variables *x* et *t* ne contiennent pas de valeur (pour *n* on prend le symbole dans la table de caractères [ctrl] [∞β°] suivi de 1).
  Ainsi, si *t* contient la valeur 0, les termes $\sin(x)$ seront remplacés par 0 et les termes $\cos(x)$ par 1.
  Il est donc prudent de faire un **clearAZ** ([menu] [1] [4]).
- Seules les équations n'utilisant que des termes en $\cos(x)$ et $\sin(x)$ pourront être résolues par cette fonction.
  Il n'est par exemple pas possible de résoudre $1+\cos\left(\frac{x}{2}\right)=0$.
- De plus, le changement de variable $t=\tan\left(\frac{x}{2}\right)$ n'est pas toujours valide. Lorsque $x=(2n+1)\pi$ est solution de l'équation, cette solution ne sera pas correctement prise en compte.
  Le problème se pose lors de la résolution de $1+\cos(x)=0$.

Avec la première version de **tzeros**, toutes ces vérifications sont laissées à l'utilisateur. Voici à présent une version plus robuste de ce programme, permettant de mieux traiter certaines de ces situations comme le montrent l'écran suivant.




```
Define LibPub tzeros (ex)=Func
©ex : zeros d'une expr. trig.
Local y,s,a
If getType(t) ≠ "NONE"
Return "DelVar t "
y:=tExpand(ex)
a:=y|x=π
y:=y|cos(x)=(1−t^2)/(1+t^2) and sin(x)=2*t/(1+t^2)
y:=getNum(y)
If inString(string(y),"x")>0
Return "Echec conversion"
s:=2*tan⁻¹(zeros(y,t))+2*n1*π
If a=0 then
 augment({(2*n1+1)*π},s)
else
 s
EndIf
EndFunc
```

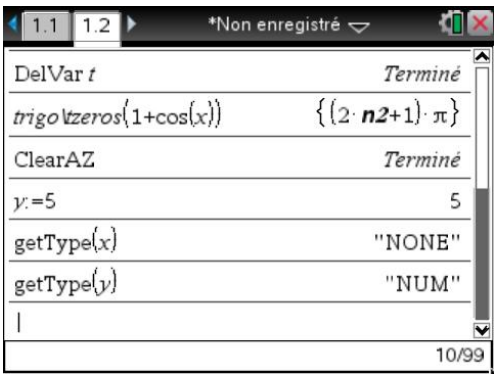
3.7 *Quelques explications techniques*

Si vous êtes un utilisateur débutant, vous pourrez revenir sur ce paragraphe lorsque vous aurez acquis les connaissances de base sur la programmation de la TI-Nspire CAS, voir le **chapitre 14** de ce livre.

- Les deux lignes

  ```
  If getType(t) ≠ "NONE"
  Return "DelVar t "
  ```

  permettent de tester s'il y a une valeur dans la variable *t*. Si c'est le cas, la fonction retourne la chaîne de caractères **"DelVar t"**.

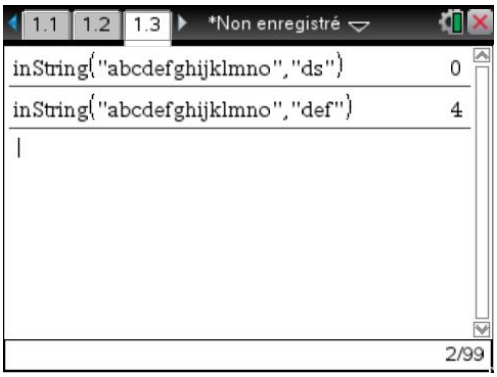


☞ *Si vous êtes déjà habitué à la programmation avec des variables locales, vous aurez peut-être pensé qu'il serait possible de déclarer t comme variable locale, afin d'éviter tout problème de ce type. Cela n'est pas possible sur la* TI-Nspire CAS *: seules des variables locales ayant préalablement reçu une valeur peuvent être utilisées dans une expression calculée par un programme ou une fonction.*

- Les deux lignes

```
If inString(string(y),"x")>0
Return "Echec conversion"
```

permettent de tester la présence éventuelle de la variable x dans l'expression obtenue après remplacement des termes en $\cos(x)$ et $\sin(x)$ en fonction de $t=\tan(x/2)$. Si x est encore présent, la fonction retourne la chaîne de caractères **"Echec conversion"**. Pour cela, on cherche si la chaîne de caractères **"x"** est présente dans la chaîne de caractères obtenue par la fonction **string** à partir de la variable **y** contenant le résultat de la substitution. La fonction **inString** permet d'effectuer ce travail : **inString**(*ch*1,*ch*2) retourne la position de *ch*2 dans *ch*1, ou 0 si *ch*2 ne se trouve pas dans *ch*1.



- Pour terminer, à la fin de la fonction, on ajoute les solutions $x=(2n+1)\pi, n\in\mathbb{Z}$ dans le cas où l'expression est nulle en $x=\pi$. Le calcul de la valeur en ce point, et sa mémorisation dans la variable **a** sont faits par l'instruction **a:=y|x=π** située vers le début de la fonction. L'instruction **augment** permet de rajouter cette solution dans la liste de solutions obtenue au préalable par la fonction **zeros**.

Exercices

1 Équation avec des complexes

Déterminer les points fixes de l'application $z \mapsto \dfrac{\overline{z}}{1+z}$.

2 Programmation linéaire

Une société fabrique et vend deux produits X et Y dont les prix de vente sont respectivement de 27€ et 23€ par unité. Les prévisions de ventes mensuelles sont de 1050 unités du produit X et 650 du produit Y au maximum. La fabrication est assurée sur trois types de machines M_1, M_2 et M_3. Le temps d'occupation des machines par unité de produit étant donné par le tableau suivant :

	X temps en heures	Y temps en heures
M_1	19	31
M_2	5	4
M_3	5	23

Le temps total disponible par mois sur chaque machine est de :

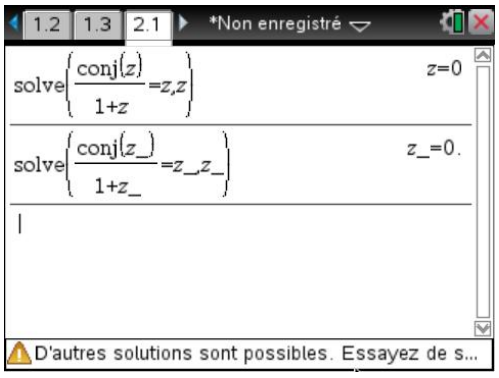
M_1	M_2	M_3
29450 h	6000 h	16100 h

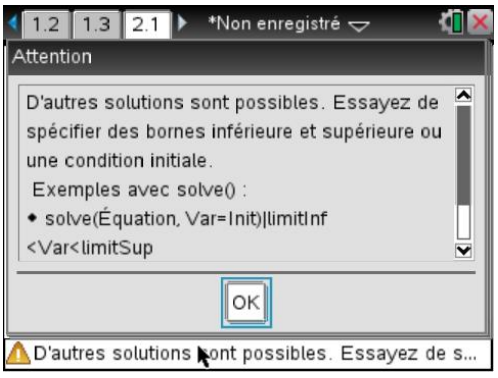
Déterminer le programme de fabrication rendant maximal le chiffre d'affaire.

Solution des exercices

1 Équation avec des complexes

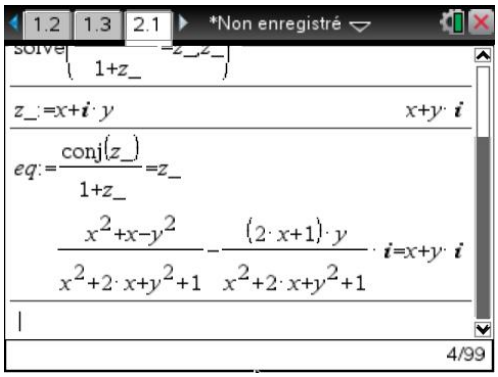
Une résolution directe n'est pas possible :

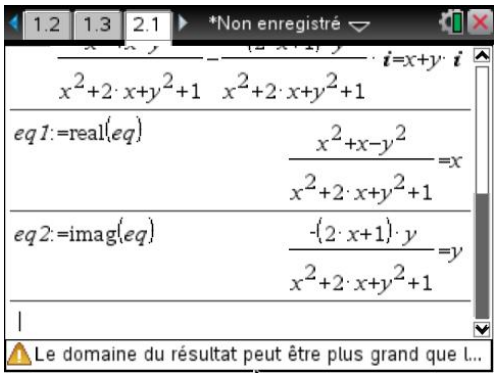


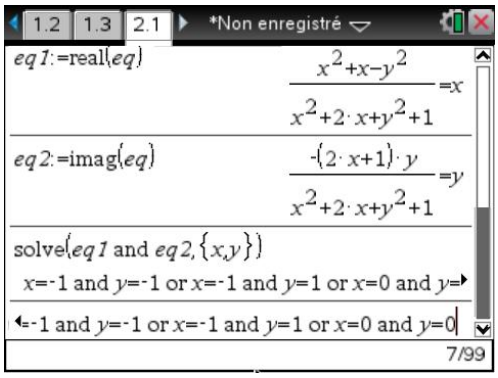


Dans le premier cas, on a obtenu la valeur 0, correspondant en fait à la résolution de l'équation $\dfrac{z}{1+z} = z$, dans le second cas, $\overline{z}$ n'est pas simplifié en z, mais la calculatrice ne parvient pas à résoudre l'équation de manière formelle (rappel le tiret bas s'obtient à l'aide des touches ctrl ␣).

La solution affichée comporte un point décimal, ce qui montre qu'elle a été obtenue à l'aide d'un calcul approché, et on peut lire un message d'avertissement en bas de l'écran, indiquant qu'il y a peut-être d'autres solutions... Pour résoudre cet exercice, il faut en fait utiliser la méthode décrite dans le chapitre précédent. Elle consiste à mémoriser $x+iy$ dans *z*, puis à résoudre le système d'équations obtenu en considérant les parties réelles et les parties imaginaires de l'équation à résoudre.







On obtient donc $S=\{0,-1+i,-1-i\}$.

2 Programmation linéaire

Si *x* représente le nombre d'unité X et *y* le nombre d'unité Y, le chiffre d'affaire est donné par :

$$f(x,y)=27x+23y$$

Le polygone des contraintes est donné par :

- contraintes de prévision de ventes :

$$\begin{cases} 0\le x\le 1050 \\ 0\le y\le 650 \end{cases}$$

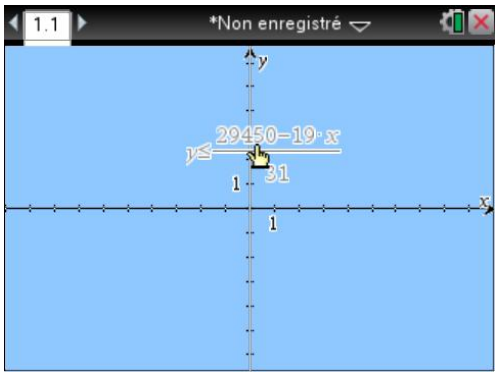
- contraintes de production :

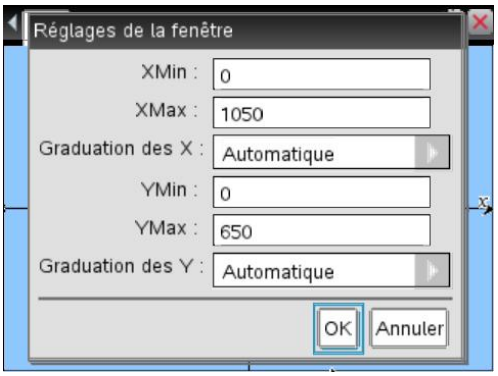
$$\begin{cases} M_1: 19x+31y\le 29450 \\ M_2: \ \ 5x+4y\le 6000 \\ M_3: 5x+23y\le 16100 \end{cases}$$

On représente le polygone des contraintes.

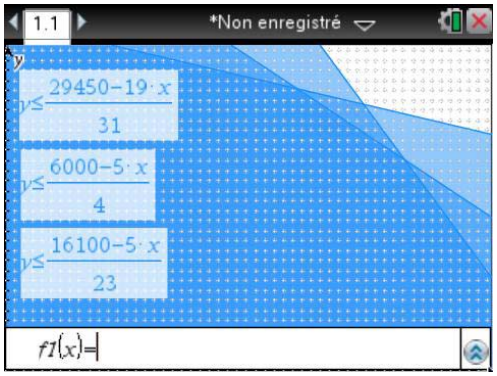
Il est possible de définir l'inéquation dans une zone de texte comme on l'a vu dans le paragraphe Inéquations, mais aussi dans la ligne d'édition. Il suffit d'effacer *f1*(*x*) et de le remplacer par une inéquation de la forme $y\le ax+b$, ou $y>ax+b$.

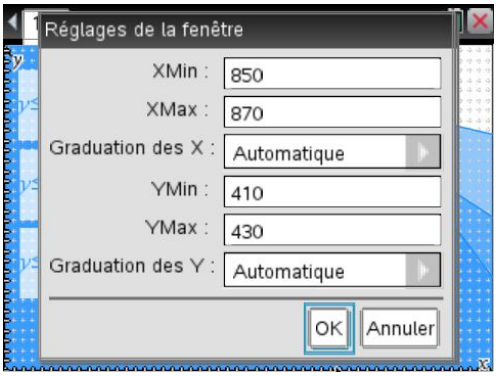
On procède ainsi pour les trois inéquations de contraintes de production, puis on choisit la fenêtre correspondant aux contraintes de prévision de ventes.

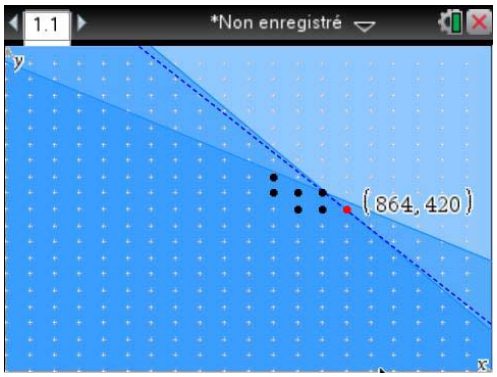




On affiche la grille, on effectue un zoom.







On obtient un chiffre d'affaire maximum pour $x = 864$ et $y = 420$, ce point se trouve sur l'isoquante la plus éloignée de l'origine, ce qui donne 32988€. Les temps d'occupation des machines seront respectivement de :

29436 heures pour M_1

6000 heures pour M_2

13980 heures pour M_3