Chapitre 9

Calcul matriciel

L'objectif de ce chapitre est de présenter l'utilisation de TI-Nspire CAS dans le domaine du calcul matriciel et de montrer comment il est possible d'en étendre les fonctionnalités.

Sommaire

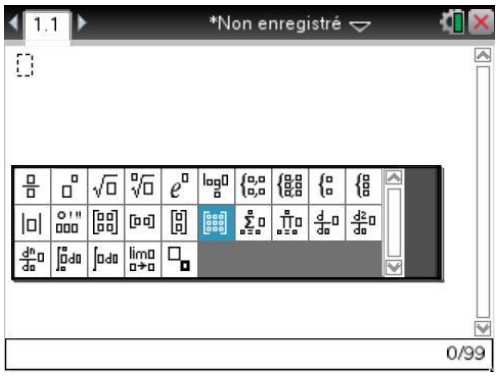
Philippe Fortin (Lycée Louis Barthou – Pau) / Roland Pomès (Lycée René Cassin – Bayonne)

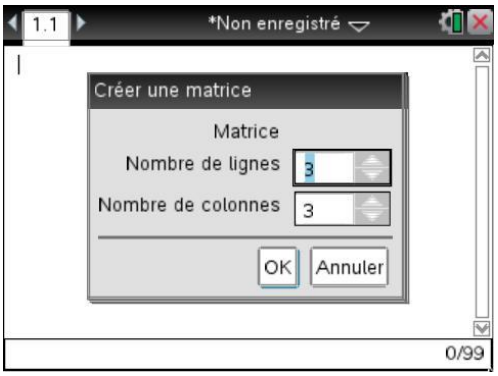
1. Présentation

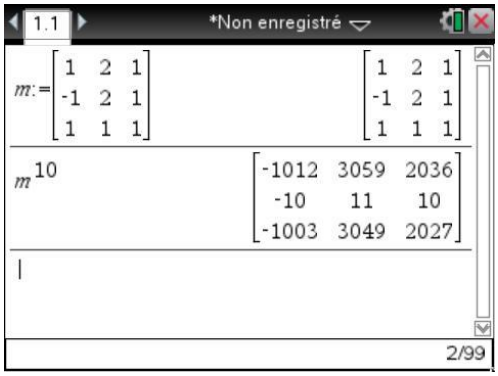
La TI-Nspire CAS permet d'effectuer directement (sans recours à l'écriture de programme) certains calculs matriciels assez complexes. Il est par exemple possible d'effectuer le calcul exact des puissances d'une matrice, et dans certains cas, le calcul approché des exponentielles.

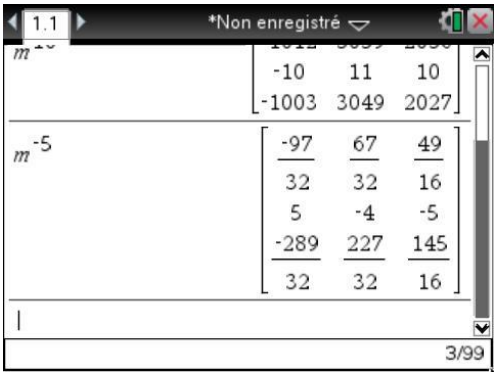
La façon la plus simple pour définir une matrice est d'utiliser l'un des modèles disponibles en appuyant sur [▦] :

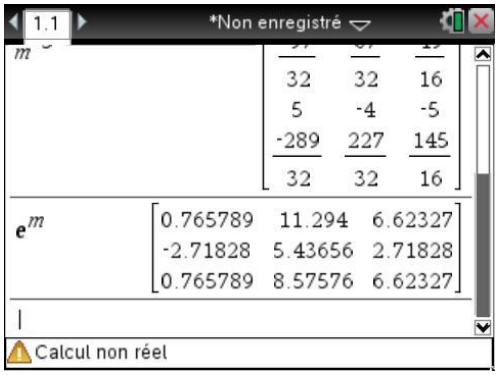
- Matrice 2×2.
- Vecteur ligne de dimension 2 (matrice 1×2).
- Vecteur colonne de dimension 2 (matrice 2×1).
- Matrice de taille quelconque. On obtient ensuite une boite de dialogue permettant de choisir le nombre de lignes et de colonnes.





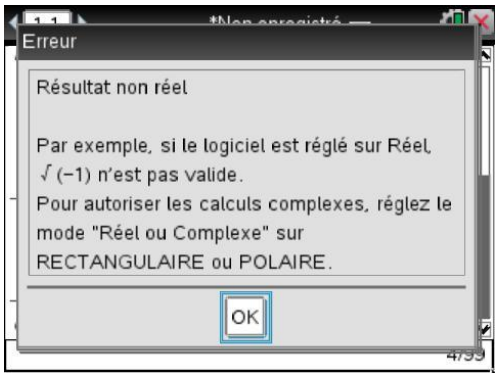


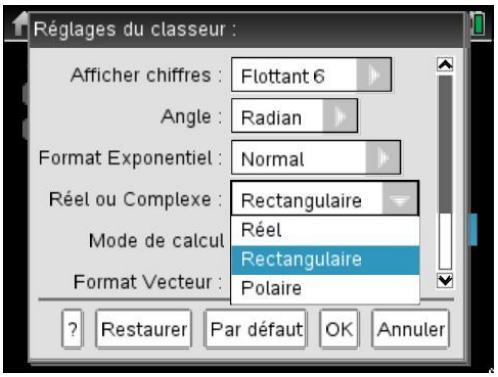




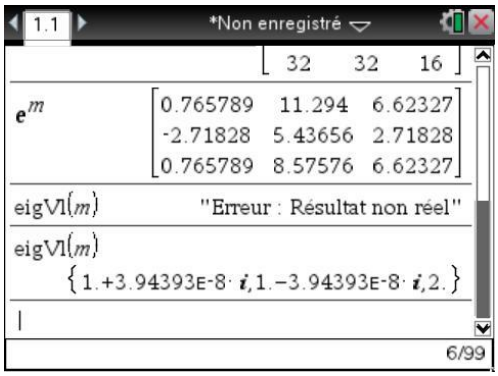
☞ *Attention au message en bas de l'écran indiquant que certaines étapes du calcul ont été faites dans l'ensemble des nombres complexes. Dans la version 5.4 le triangle jaune apparait au niveau de la saisie et il faut cliquer dessus pour que le message s'affiche.*

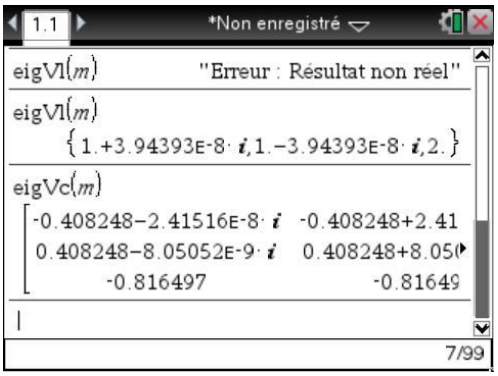
On peut aussi obtenir l'expression approchée des valeurs ou vecteurs propres (dans menu 7 B) :





☞ *Un calcul en mode réel peut entraîner le message d'erreur ci-dessus, on est convié à passer en mode complexe.*





Les fonctions **eigVl** et **eigVc** sont des fonctions utilisant des algorithmes numériques, qui permettent d'obtenir des *valeurs approchées* des valeurs propres et des vecteurs propres. Ces algorithmes travaillent dans ℂ. Du fait des erreurs liées aux calculs numériques, il est possible que des valeurs qui devraient être réelles soient obtenues sous la forme de nombres complexes ayant une partie imaginaire très petite. C'est ce qui est arrivé dans l'écran ci-dessus.

Nous allons voir dans ce chapitre qu'il est également possible d'utiliser d'autres fonctions, permettant de résoudre différents problèmes classiques sous forme symbolique :

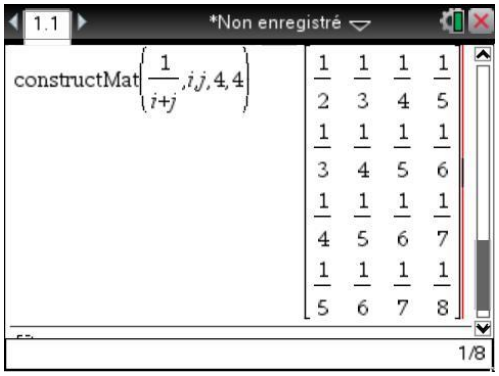
- Recherche du rang d'une matrice.
- Recherche du noyau d'un endomorphisme.
- Recherche de l'expression symbolique des valeurs et vecteurs propres.
- Diagonalisation d'un endomorphisme.
- Réduction d'un endomorphisme non diagonalisable.
- Calcul des puissances symboliques d'une matrice.
- Calcul de l'exponentielle d'une matrice.

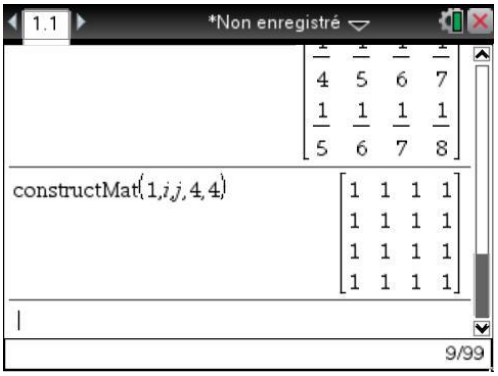
Les fonctions et programmes que nous allons rencontrer et décrire dans ce chapitre font partie de la bibliothèque **linalgcas** téléchargeable avec ce document, voir également le **chapitre 15**.

2. Création de matrices

Comme nous l'avons vu dans le paragraphe précédent, on peut utiliser différents modèles prédéfinis. On peut aussi utiliser la syntaxe `[1,2;3,4]` le séparateur `;` (?!▸) indiquant le changement de ligne.

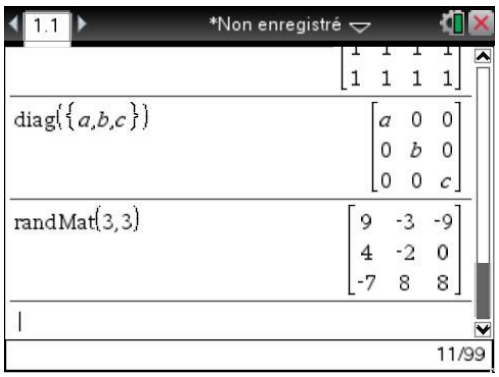
Le sous-menu **Créer** du menu **Matrice & vecteur,** accessible par menu 7 1, contient différentes fonctions de création de matrices. En particulier, la fonction **constructMat** permet de construire les matrices dont le terme général a_{ij} est défini par une expression.

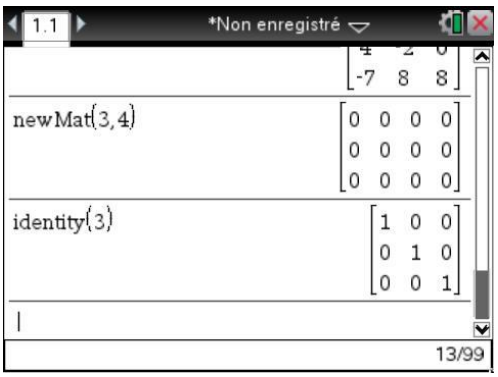




Autres fonctions :

- **diag**(*liste*) **:** matrice diagonale. La diagonale est définie par une liste (entre { }).
- **randMat**(*n*, *p*) : matrice aléatoire, ayant *n* lignes et *p* colonnes.
- **newMat**(*n*, *p*) : matrice nulle, ayant *n* lignes et *p* colonnes.
- **identity**(*n*) : matrice unité *n*× *n*.





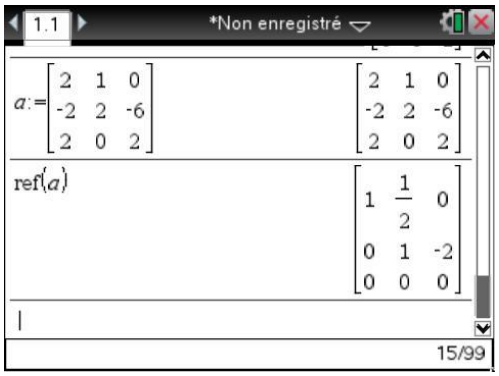
3. Recherche du rang d'une matrice

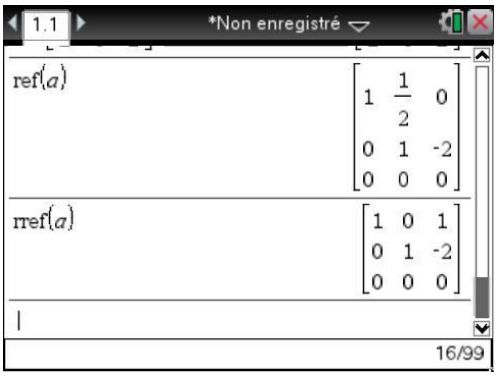
3.1 *Étude directe à partir d'une réduite de Gauss*

Considérons par exemple la matrice $A = \begin{bmatrix} 2 & 1 & 0 \\ -2 & 2 & -6 \\ 2 & 0 & 2 \end{bmatrix}$.

Pour obtenir le rang de cette matrice, il suffit d'effectuer une réduction de Gauss de cette matrice (**ref**), et de compter le nombre de lignes non nulles.

☞ *On peut aussi utiliser la réduite de Gauss-Jordan (**rref**).*





Dans cet exemple, on peut voir qu'il y en a deux. Le rang est donc égal à 2.

3.2 *Fonction de calcul du rang*

Il est facile d'écrire une fonction automatisant ce décompte. L'idée est d'ajouter des nombres n_i, égaux à 1 si la ligne n°i est non nulle, et égaux à 0 dans le cas contraire. Pour tester si la ligne n°i est nulle, il suffit de tester la valeur de sa norme. Ce qui conduit à l'écriture de la fonction suivante.

```
Define LibPub rank(m)=Func
©m: rang de m
Local n
n:= rref(m)
∑(when(norm(n[i])=0,0,1),i,1,rowDim(n))
EndFunc
```

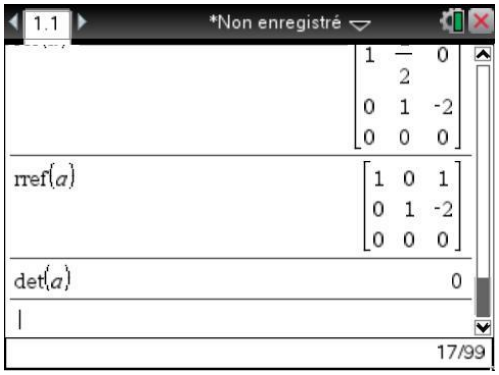
- *La fonction* **when** *s'utilise sous la forme*
  when(*condition, résultat-si-vrai, résultat-si-faux, résultat-si-l'on-ne-sait-pas*).
- *Si* **n** *est une matrice,* **n[i]** *correspond à la i-ème ligne.* **norm(n[i])** *sera égal à 0 si et seulement si cette ligne est nulle.* **rowDim(n)** *correspond au nombre de lignes de la matrice.*

Reportez-vous au **chapitre 14** pour une découverte de la programmation sur TI-Nspire CAS, et au **chapitre 15** pour plus d'information sur l'utilisation des commandes **Define LibPub**, **Define LibPriv**.

4. La recherche du noyau d'un endomorphisme

Considérons à nouveau la matrice $A = \begin{bmatrix} 2 & 1 & 0 \\ -2 & 2 & -6 \\ 2 & 0 & 2 \end{bmatrix}$.

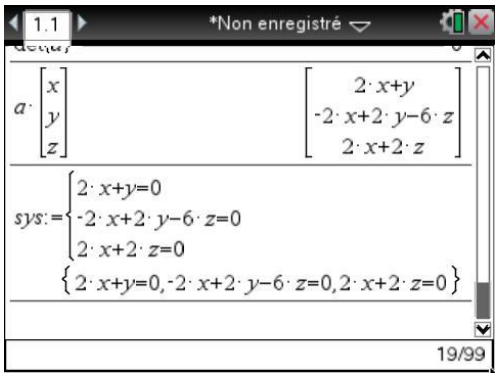
Même sans utiliser ce qui précède, on peut vérifier que cette matrice n'est pas inversible. Il suffit de calculer le déterminant de la matrice.

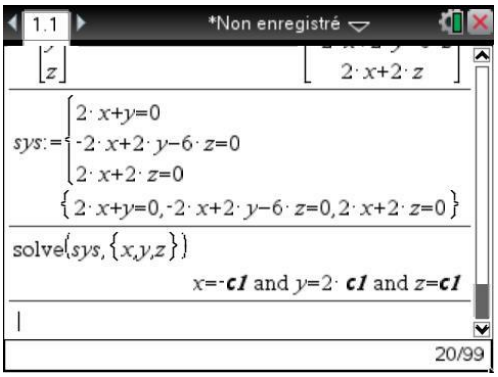


Comment obtenir le noyau de l'application linéaire associée ?

4.1 *Recherche du noyau avec la fonction solve*

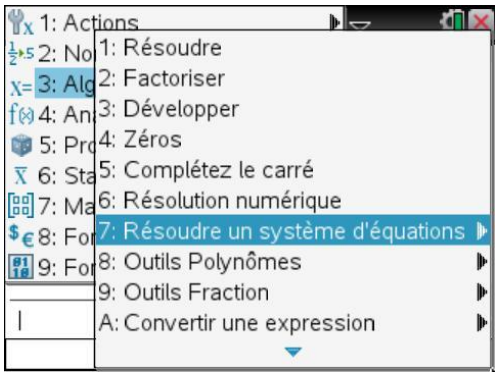
Une première méthode consiste à résoudre le système d'équations définissant le noyau avec la fonction **solve**.
Le système est écrit à l'aide du modèle {, disponible en appuyant sur [ ].

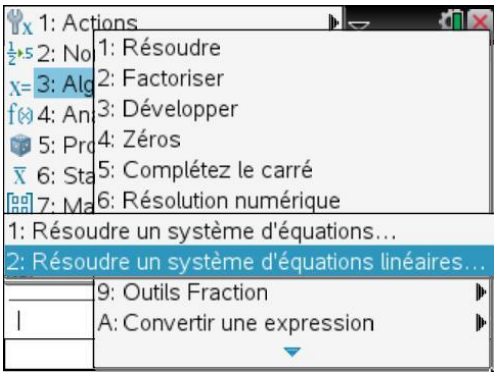


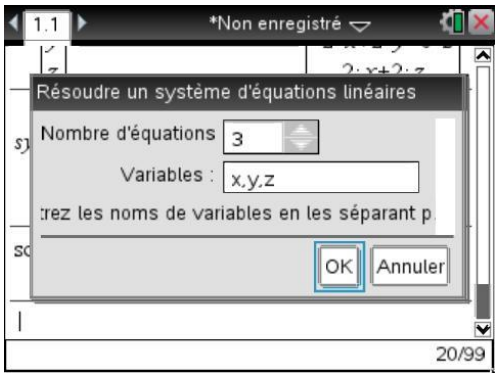


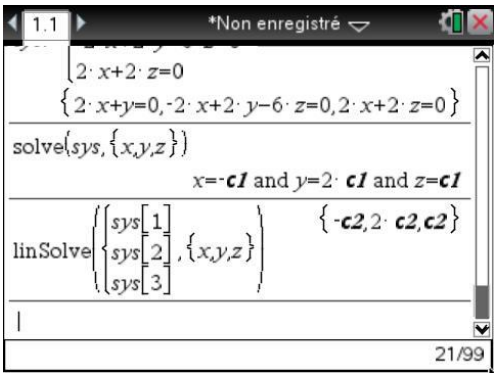
Les solutions sont donc du type $(-z, 2z, z)$, avec z quelconque. Le noyau est donc de dimension 1, et il est engendré par le vecteur de coordonnées $(-1, 2, 1)$.

On peut également utiliser l'assistant pour résoudre le système d'équations linéaires qui fait appel à la fonction **linSolve**.





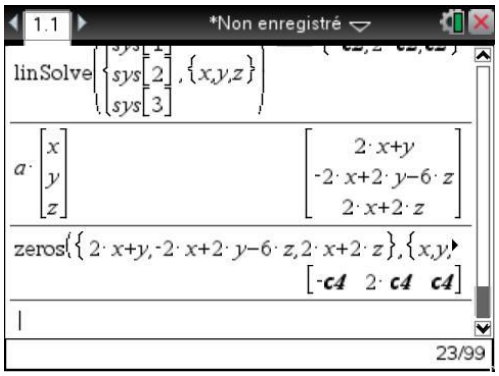




4.2 *Recherche du noyau avec la fonction zeros*

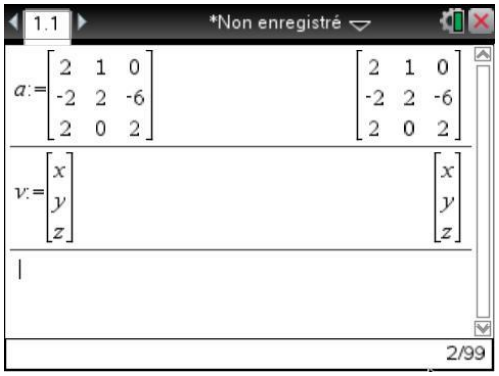
On peut aussi choisir d'utiliser la fonction **zeros**, qui permet d'avoir un résultat plus compact :

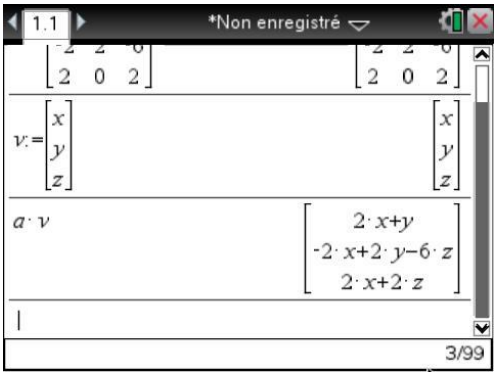
```
zeros({2x+y,-2x+2y-6z,2x+2z},{x,y,z})
```



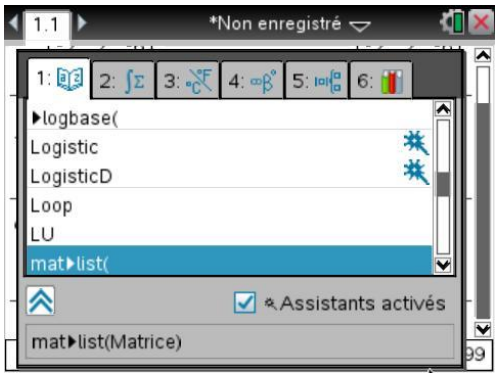
Il est possible de construire automatiquement la liste des équations à résoudre.

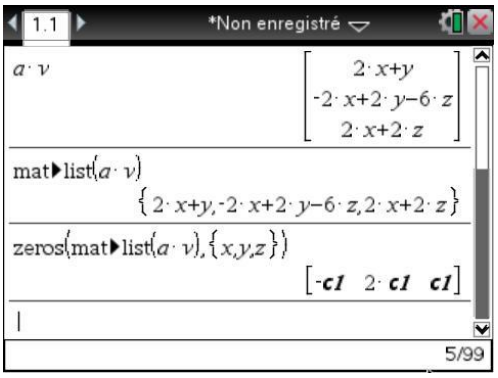
Pour cela, on définit un vecteur quelconque, et on fait le produit de la matrice par ce vecteur :





La fonction de conversion **mat▸list** (/ 1 M) permet ensuite de récupérer la liste des composantes, et il est ensuite possible d'utiliser la fonction **zeros** :





4.3 *Fonction de recherche des vecteurs du noyau*

Il est possible d'automatiser ce qui vient d'être fait en utilisant deux fonctions.

Attention, le texte de ces deux fonctions utilise des méthodes un peu sophistiquées, vous pouvez les ignorer dans un premier temps.

```
Define LibPriv vect(n)=Func
©n: vecteur de dim n
Local i,v
v:=newMat(n,1)
For i,1,n
v[i,1]:=expr("θ"&string(i))
EndFor
v
EndFunc
```

On commence par construire une matrice colonne de *n* lignes, ne contenant que des 0. Ensuite, une boucle construit les variables **θ1**, **θ2**, … qui sont placées dans ce vecteur. On utilise pour cela plusieurs fonctions de manipulation des chaînes de caractères.

1. La fonction **string** est utilisée pour convertir le contenu de la variable *i* : 1, 2, … en chaîne de caractères : "1", "2", …
2. En utilisant l'opérateur de concaténation **&**, on colle cette chaîne à "**θ**" pour obtenir "**θ1**", " **θ2**", …
3. La fonction **expr** permet de fabriquer les noms de variables **θ1**, **θ2**, … à partir de ces dernières chaînes de caractères.

Voici à présent le texte de la fonction permettant d'obtenir les vecteurs du noyau par la méthode décrite dans le paragraphe précédent :

```
Define LibPub vker(m)=Func
© mat: vect. de ker(mat)
Local i,n,v,leq,linc
n:=colDim(m)
v:=vect(n)
leq:=mat▸list(m*v)
linc:=mat▸list(v)
expr("zeros("&string(leq)&","&string(linc)&")")
EndFunc
```

La fonction **vect** construit un vecteur quelconque de dimension donnée. La fonction **vker** utilise ce vecteur pour déterminer le noyau en utilisant la méthode précédente.

☞ ***vect*** *doit être définie dans la même activité que* ***vker*** *pour que cette dernière puisse fonctionner (voir le* ***chapitre 15*** *à ce sujet).*

La dernière ligne de cette fonction est pour le moins ésotérique. Elle permet de résoudre un problème qui n'a rien d'évident.

Si une variable, par exemple **leq**, contient la liste des équations, par exemple **{2x+y,x−y}** et si une autre variable, **linc**, contient la liste des inconnues, par exemple **{x,y}**, alors

```
zeros(leq,linc}
```

n'est pas équivalent à

```
zeros({2x+y,x−y},{x,y})
```

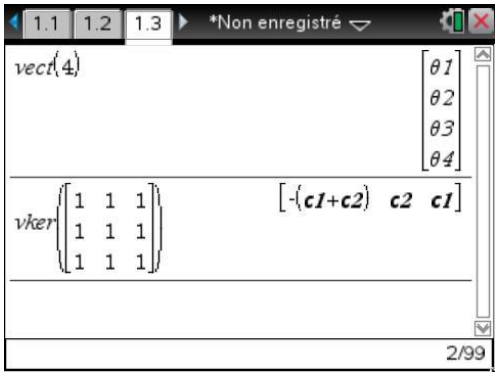
Ceci provient d'un mécanisme d'évaluation un peu particulier de la fonction **zeros**. Comme nous l'avons vu dans l'exemple du paragraphe précédent, il est possible de mémoriser une liste d'équations dans une variable, par contre, il n'est pas possible de faire de même avec la liste des inconnues.

Le seul moyen de résoudre le problème consiste à construire la chaîne de caractères :

```
"zeros({2x+y,x−y},{x,y})"
```

à partir du contenu des variables **leq** et **linc**, puis à exécuter l'instruction contenue dans cette chaîne à l'aide de la fonction **expr**. C'est exactement ce que fait notre dernière ligne…

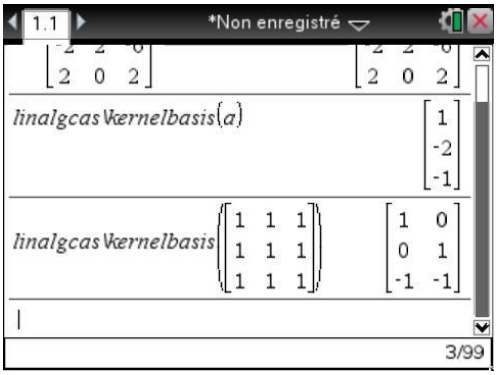
Voici un exemple d'utilisation de ces deux fonctions :



☞ *La fonction* **vker** *correspond à la fonction* **kernelvectors** *de la bibliothèque* **linalgcas**, *jointe à ce document.*

4.4 *Fonction de recherche d'une base du noyau*

Nous allons maintenant écrire une fonction qui sera capable de déterminer directement une base du noyau d'une application linéaire définie par sa matrice. Voici deux exemples illustrant l'utilisation de cette fonction **kernelbasis** :



4.5 *Texte de la fonction kernelbasis*

```
Define LibPub kernelbasis(m)=Func
©mat: noyau de mat nxn
Local a
If det(m)≠0 Then
[0]
Else
a:=rref(augment(mᵀ,1+0*m))
(subMat(a, rank(m)+1,rowDim(m)+1))ᵀ
EndIf
EndFunc
```

4.6 *Quelques explications sur la fonction kernelbasis*

Les explications qui suivent nécessitent quelques connaissances de base sur l'interprétation des opérations sur les lignes d'une matrice en terme de produits matriciels, et sur le calcul de produit par blocs. Si vous ne connaissez pas encore ces notions, n'hésitez pas à simplement laisser de côté la suite de ce paragraphe.

Dans ce qui suit on parle du "noyau d'une matrice", il s'agit d'un raccourci pour désigner le noyau de l'endomorphisme associé à cette matrice.

Pour chercher la réduite de Gauss d'une matrice, on effectue des opérations élémentaires sur les lignes de la matrice. Cela revient à faire un produit, à gauche, par une matrice inversible.

Soit *A* une matrice $n \times n$, et I_n la matrice identité de même dimension.

Si l'on fait une réduction de ce type sur une matrice du type $M = \begin{array}{cc} A & I_n \end{array}$, matrice $n \times 2n$, on obtient $G = T \cdot \begin{array}{cc} A & I_n \end{array} = \begin{array}{cc} T \cdot A & T \cdot I_n \end{array} = \begin{array}{cc} T \cdot A & T \end{array}$.

Par ailleurs, en raison de la nature d'une réduite de Gauss, la matrice *G* sera du type $G = \begin{bmatrix} P & Q \\ O & R \end{bmatrix}$, avec *P et Q* matrices triangulaires $r \times n$, *O* matrice nulle $(n-r) \times n$, *R* matrice $(n-r) \times n$.

En identifiant avec ce qui précède, on a
$T = \begin{bmatrix} Q \\ R \end{bmatrix}$ et $G = \begin{bmatrix} Q \\ R \end{bmatrix} \cdot \begin{array}{cc} A & I_n \end{array} = \begin{bmatrix} Q \cdot A & Q \\ R \cdot A & R \end{bmatrix} = \begin{bmatrix} P & Q \\ 0 & R \end{bmatrix}$.

En particulier $R \cdot A = 0$ et donc, en transposant, ${}^tA \cdot {}^tR = 0$.
La matrice ${}^tR$ est une matrice $n \times (n-r)$, elle est extraite de ${}^tT$ qui est inversible. Les $n-r$ vecteurs colonnes de cette matrice forment donc une famille libre.

De plus, ${}^tA \cdot {}^tR = 0$, et donc pour tout vecteur colonne V_j de ${}^tR$, ${}^tA \cdot V_i = 0$.

On a ainsi trouvé $n-r$ vecteurs V_j, formant une famille libre, et tous dans le noyau de ${}^tA$ qui est précisément de dimension $n-r$.

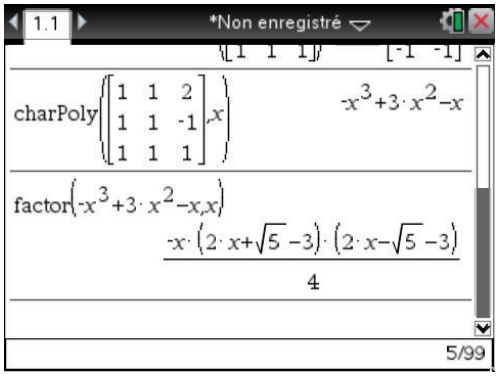
En conclusion, la réduction de Gauss de la matrice $M = \begin{array}{cc} A & I_n \end{array}$ sous la forme $G = \begin{bmatrix} P & Q \\ O & R \end{bmatrix}$ permet de trouver une base du noyau de ${}^tA$: il suffit d'extraire *R* et de transposer cette matrice.

Pour obtenir une base du noyau de *A*, il suffit de partir de $M = \begin{array}{cc} {}^tA & I_n \end{array}$ comme on le fait dans la fonction **kernelbasis**.

5. Éléments propres

5.1 *Valeurs propres d'une matrice*

Pour calculer les valeurs exactes des valeurs propres d'une matrice, on doit "simplement" rechercher les racines du polynôme caractéristique. Ce polynôme s'obtient à l'aide de la fonction **charPoly** qui correspond au calcul de $\det(M - xI_n)$.



Voici maintenant deux courtes fonctions permettant de résoudre le problème de la recherche des valeurs propres.

La fonction **valp** recherche les valeurs propres réelles :

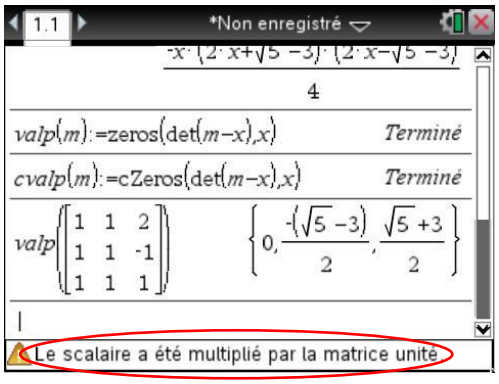
```
Define LibPub valp(m)=zeros(det(m-x),x)
```

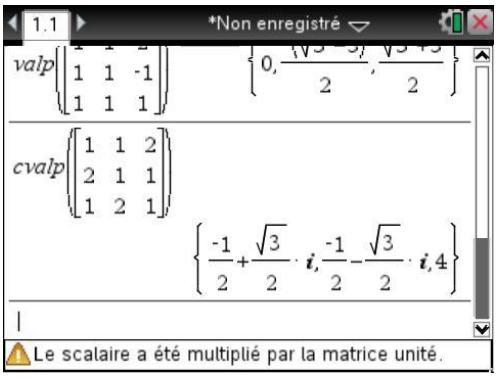
La fonction **cvalp** traite également les valeurs propres complexes :

```
Define LibPub cvalp(m)=cZeros(det(m-x),x)
```

☞ *Lors de l'addition entre une matrice de $M_n(\mathbb{K})$ et un scalaire λ, le scalaire est automatiquement identifié à la matrice λI_n.*

Vous trouverez dans la bibliothèque **linalgcas** les fonctions **eigenvals** et **ceigenvals** ayant les mêmes fonctionnalités que les fonctions ci-dessus.



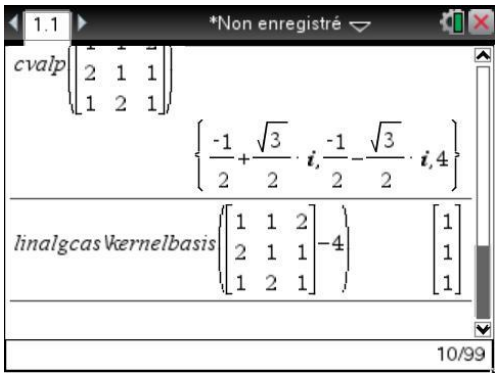


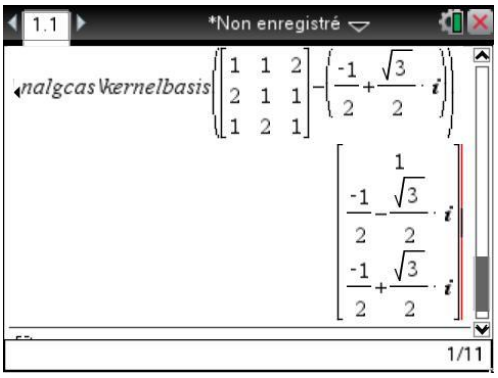
☞ *Le logiciel de la TI-Nspire CAS comme Maple autorise l'écriture $m - x$, où m est une matrice carrée et x un scalaire et l'interprète comme $m - x.I_n$, I_n matrice unité. C'est la signification du message au bas de l'écran.*

5.2 *Vecteurs propres d'une matrice*

Pour déterminer les vecteurs propres d'un endomorphisme f défini par une matrice M, il suffit d'utiliser la fonction **kernelbasis** pour déterminer une base de chaque $E_\lambda = \ker(f - \lambda Id)$.

Voici par exemple une base de deux espaces propres associés à la dernière matrice :





6. Diagonalisation

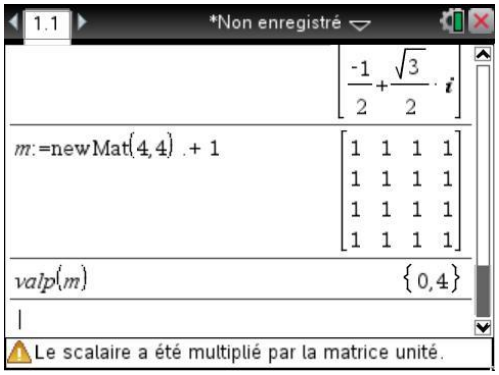
6.1 *Étude directe de la diagonalisation*

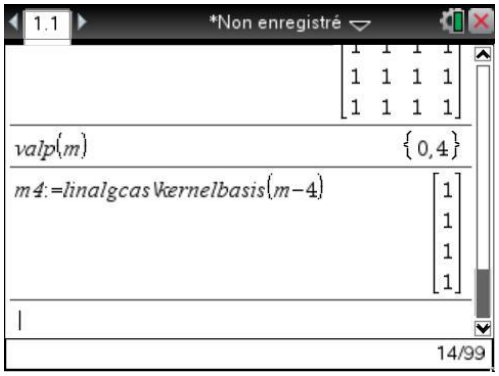
Tout est en place pour résoudre le problème de la diagonalisation des matrices.

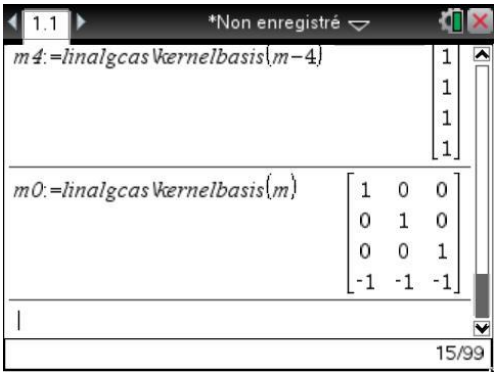
Il suffit de rechercher les valeurs propres, puis les espaces propres associés.

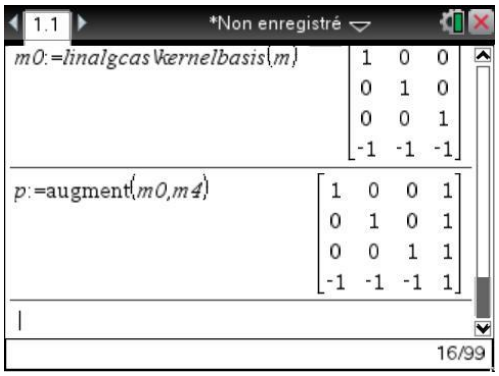
Concrètement, on doit “concaténer” les matrices donnant les bases de chaque espace propre pour obtenir la matrice de passage dans le cas où la matrice est effectivement diagonalisable.

En voici un exemple :









☞ *Sur le premier écran,* **newMat** *permet de créer une matrice nulle de la taille indiquée, l'opérateur* **.+** *permet d'ajouter* 1 *à chaque élément de la matrice, et donc d'obtenir la matrice avec tous les coefficients égaux à* 1.
On aurait également pu utiliser l'instruction **fill** *ou directement* **constructMat** *pour obtenir ce résultat.*

☞ *Sur le quatrième écran, la fonction* **augment** *permet de créer une matrice à partir de deux autres matrices ayant un même nombre de lignes. Pour superposer deux matrices ayant le même nombre de colonnes, on utilisera la fonction* **colAugment**.

6.2 *Utilisation d'un programme de diagonalisation*

La diagonalisation est effectuée automatiquement par le programme **diagonalization**[1] de la bibliothèque **linalgcas** (voir **chapitre 15**).

1. Ce programme affiche la première valeur propre λ_1.
2. Puis, il affiche la matrice obtenue en cherchant une réduite de Gauss de $M - \lambda_1 I_n$, ce qui est utile lors d'une résolution manuelle pour déterminer le rang de cette matrice, la dimension de l'espace propre associé, et les éléments de cet espace propre.
3. On obtient ensuite la matrice regroupant les vecteurs formant une base de l'espace propre.

Les opérations 1, 2, 3 sont répétées pour chaque valeur propre.

4. Le programme affiche le cas échéant un message indiquant que la matrice est diagonalisable.
5. Dans ce cas il affiche *P* et *D*, puis le polynôme minimal de la matrice. Ces éléments sont mémorisés sans les variables **θP**, **θD** et **θpol** en vue d'un éventuel usage ultérieur.

[1] Il s'agit de l'orthographe anglaise. Les noms des programmes ou fonctions de la bibliothèque de programmes **linalgcas** sont en anglais, comme pour toutes les fonctions intégrées à TI-Nspire CAS. Par contre, la langue des messages affichés lors de l'utilisation de cette bibliothèque s'adapte automatiquement au choix de la langue fait dans le menu **Réglages du système**.

6.3 *Exemple d'utilisation*

On recherche ici la diagonalisation de $M = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$