Qu'est ce que Maple™ ?

Maple™, est dit un logiciel de calcul formel, en opposition avec calcul numérique (ou scientifique) qui a aussi ses représentants tel Matlab™ ou Scilab. Il est édité par la société canadienne MapleSoft qui vient d'en sortir la version 14. Maple™ est capable de faire des calculs de façon symbolique sans se soucier de la valeur numérique des variables utilisées, de la même manière qu'un mathématicien manipule des équations sans au préalable imposer des valeurs données à ses variables.

Maple™ dispose, depuis la version 7, de deux interfaces graphiques : l'interface historique renommée Maple Classic™, et une plus récente totalement réécrite en Java. Ce qui a permis à Maple™ d'être porté avec un moindre effort vers les plateformes moins courantes (mais majoritaires dans les milieux universitaires) de type Unix, mais aussi MacOS X. L'interface Classic est disponible pour MS Windows™ et continue à être proposée, mais sans vraiment évoluer, dans les versions récentes, et pour Linux à travers l'éxécutable **wcmaple**. Elle n'est par contre pas fournie pour MacOS X. Pour la nouvelle interface le programme d'installation intègre une machine virtuelle Java™ et en général l'utilisateur n'a pas besoin d'en installer une (qui de toute façon ne sera pas utilisée, à moins d'intervenir dans ce sens). Le logiciel intègre un système d'aide en ligne qui peut s'avérer une source d'apprentissage inéstimable et une référence complète pour tout ce qui touche à son utilisation. Il propose en outre depuis la version 7, une technologie appelée Maplet™, qui permet d'exporter de façon transparente un programme Maple™ en une applet Java, qui de cette manière peut être par exemple diffusée sur une page web. Il a aussi depuis un certain temps acquis des capacités de logiciel tableur et peut gérer en import/export des fichiers MS Excel™. Les Maplets et les fonctionnalités tableur ne sont actives que dans l'interface Java™.

Les élèves des classes préparatoires peuvent l'utiliser comme une simple (super)calculatrice pour le calcul, disons de limites, de sommes de séries, de développement limités, d'intégrales et de primitives, de déterminant, valeurs et vecteurs propres de matrices... Il permet de résoudre de façon simple diverses sortes d'équations (système d'équations algébriques, équations de récurrence, équations différentielles). Et il a des capacités graphiques, tel le tracé de graphes de fonctions ou de surfaces dans l'espace.

En outre, Maple™ intègre son propre langage de programmation, ce qui permet d'étendre ses capacités en définissant soit même de nouvelles fonctions. Dans ce sens il peut très bien être utilisé comme un environnement d'initiation à la programmation sans les contraintes liées aux déclarations de types de variables (et donc de la gestion de la mémoire) et aux problèmes de compilation inhérents aux langages de programmation classiques.

« Cours Maple »

Algèbre linéaire avec Maple

par Sadik BOUJAIDA

Lycée Moulay Youssef – Rabat

Un cours dont l'objectif est de donner au lecteur les moyens de résoudre des problèmes de l'algèbre linéaire en s'aidant du logiciel de calcul formel Maple. Il ne s'arrête pas à la description de quelques commandes calculatoires pour avoir telle ou telle information sur une matrice, mais adopte une approche plus constructive et commence par définir les bases d'une utilisation relativement avancée du logiciel.

I. Les listes et les ensembles :

I.1 Séquences :

Une séquence maple est une suite d'objets maple quelconques, séparés par des virgules. Une séquence en elle même est un objet maple et peut donc recevoir un nom (affectation).

```
> S :=1,2,x^2+1,exp(a),diff(x*ln(x),x);
```

$$S := 1, 2, x^2+1, e^a, \ln(x)+1$$

Maple™

On peut accéder à un élément d'une séquence en utilisant les crochets [ ] et sa position dans la séquence.

```
> S[1] ;S[3] ;
```

$$1$$

$$x^2+1$$

Maple™

Pour construire des séquences on dispose de l'operateur **$** et de la procédure **seq.**

```
> k $k=1..5 ;  #construction de la sequence 1,2,3,4,5 ;
```

$$1, 2, 3, 4, 5$$

```
> x^k $k=0..5 ;
```

$$1, x, x^2, x^3, x^4, x^5$$

```
> x$3 ;      #construction de la séquence x,x,x
```

$$x, x, x$$

Maple™

Exemples d'utilisation du constructeur **$:**

```
> diff(x^5,x,x,x) ;diff(x^5,x$3) ;
```

$$60\,x^2$$

$$60\,x^2$$

Maple™

et pour **seq :**

```
> seq(x^k,k=0..5) ;
```

$$1, x, x^2, x^3, x^4, x^5$$

Maple™

La séquence vide est NULL.

```
> L :=NULL ;
```

$$L :=$$

```
> L :=L,a ;
```

$$L := a$$

Maple™

I.2 Listes :

Une liste est une séquence mise entre crochets [ ], elle correspond à la notion mathématique de famille finie d'éléments. La liste vide est [ ].Si L est une liste maple :

L[k] **op(k,L)**	désigne le $k^{ème}$ élément de L
op(L)	retourne la séquence des éléments de la liste L
nops(L)	retourne le nombre d'éléments dans L

REMARQUES

1. **op** et **nops** ne fonctionnent pas avec les séquences
2. **$** et la procédure **seq** peuvent être utilisés pour construire des listes.

Liste des polynômes de Legendre $P_n = \dfrac{d^n\left[\left(x^2-1\right)^n\right]}{dx^n}$ de 1 à 4 :

```
> L :=[seq(diff((x^2-1)^k,x$k),k=1..4)] ;
```

$$L := \left[2\,x, 12\,x^2-4, 48\,x^3+72\left(x^2-1\right)x, 384\,x^4+1152\left(x^2-1\right)x^2+144\left(x^2-1\right)^2\right]$$

```
> op(3,L) ;
```

$$48\,x^3+72\left(x^2-1\right)x$$

```
> L[3] ;
```

$$48\,x^3+72\left(x^2-1\right)x$$

```
> nops(L) ;
```

$$4$$

Maple™

On peut directement changer la valeur d'un élément de la liste par affectation (ne marche pas avec une séquence)

```
> L[4] :=0 ;

> L ; #pour voir si L[4] a effectivement changé.
```

$$[2x, 12x^2 - 4, 48x^3 + 72\left(x^2 - 1\right)x, 0]$$

Maple™

Et pour récupérer la séquence des éléments de la liste L :

```
> S :=op(L) ;
```

$$S := 2x,\ 12x^2 - 4,\ 48x^3 + 72\left(x^2 - 1\right)x,\ 0$$

Maple™

I.3 Ensembles :

Un ensemble est une séquence placée entre accolades { }, ceci correspond à la notion effective d'ensemble fini. L'ensemble vide est { }.
La différence avec une liste est que dans un ensemble l'ordre d'entrée des éléments n'est pas respecté, les éléments en doubles sont aussi éléminés.

```
> K :={1,2,E,Pi,1} ;
```

$$K := \{1, 2, E, \pi\}$$

Maple™

Noter que le double de 1 à été éléminé et que l'ordre des éléments à changé.
Ce qui a été dit pour les listes reste valable pour les ensembles avec quelques différences de comportements.

```
> K[3] ;op(3,K) ;
```

$$\pi$$
$$\pi$$

```
> nops(K) ;
```

$$4$$

```
> op(K) ;
```

$$1,\ 2,\ \pi,\ E$$

Maple™

I.4 Conversion de types

On peut convertir un objet de type donné , en un autre d'un autre type mais avec les mêmes opérandes, une liste en un ensemble par exemple. Pour cela Maple offre la commande **convert**. Il suffit de connaitre le nom du type cible : une liste est une *list* est un ensemble est un *set* (par exemple).

```
> L :=[x,y,z] ;S :={0,0,1,a,b,c} ;
```

$$L := [x, y, z]$$
$$S := \{0, 1, a, b, c\}$$

```
> convert(L,set) ;
```

$$\{x, y, z\}$$

```
> convert(S,list) ;
```

$$[0, 1, a, b, c]$$

Maple™

Noter que pour changer le type d'un objet en une séquence, il n'est pas besoin d'utiliser **convert**, la procédure **op** suffit

```
> op(L) ;
```

$$x,\ y,\ z$$

Maple™

Une utilisation utile de cette commande est la possibilité de récupérer rapidement l'ensemble des coefficients d'une matrice.

```
> A :=matrix(2,2,[x,y,z,t]) ;
```

$$A := \begin{bmatrix} x & y \\ z & t \end{bmatrix}$$

```
> convert(A,set);
```

$$\{t, x, y, z\}$$

Maple™

d'autres exemples

```
> convert([x,y,z],`+`);
```

$$x + y + z$$

```
> convert([x,y,z],vector);
```

$$\begin{bmatrix} x & y & z \end{bmatrix}$$

Maple™

II. Expressions et procédures :

II.1 Expressions, expressions polynomiales :

Une expression maple correspond à ce que on peut qualifier d'expression mathématique faisant intervenir des variables, des constantes, des opérateurs, éventuellement des fonctions usuelles ...

```
> a :=x^2+1;
```

$$a := x^2 + 1$$

```
> b :=x*ln(x+1);
```

$$b := x \ln(x + 1)$$

```
> c :=diff(b,x$2)/a;
```

$$c := \left(2 (x+1)^{-1} - \frac{x}{(x+1)^2}\right)\left(x^2+1\right)^{-1}$$

```
> d :=int(a^2+1,x);
```

$$d := 2x + 1/5\, x^5 + 2/3\, x^3$$

Maple™

a,b,c,d sont des exemples d'expressions. Dans une expression, on peut substituer une variable à une autre en utilisant **subs**

```
> subs(x=2,b);
```

$$2 \ln(3)$$

```
> subs(x=y+1,b);
```

$$(y+1) \ln(y+2)$$

Maple™

On notera que les indeterminées d'une expression ne sont pas considérées comme des variables scalaires, mais des parametres formels au sens large du terme, et donc dans une substitution par exemple vous pouvez donner à une variable une valeur matricielle tant qu'il n'y a pas d'incompatibilité aboutissant à une erreur à l'éxécution.
Un polynôme et en particulier une expression faisant intervenir une ou plusieurs indéterminées et des constantes liées par les operateurs **+** et *****

```
> P :=sum((n+1)*x^n,n=0..5);
```

$$P := 1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5$$

```
> Q :=product(x^k-y,k=1..4);
```

$$Q := (x-y)\left(x^2-y\right)\left(x^3-y\right)\left(x^4-y\right)$$

Maple™

Coefficients d'un polynôme, en utilisant **coeff** :

```
> coeff(P,x^2);
```

$$3$$

```
> coeff(Q,y^3);
```

$$-x - x^2 - x^3 - x^4$$

```
> coeff(Q,y,3) ; #la même chose que l'instruction précédente
```

$$-x - x^2 - x^3 - x^4$$

Maple™

Pour récuperer la séquence des coéfficients (non nuls) d'un polynôme, on dispose de **coeffs** :

```
> coeffs(P,x) ;
                          1, 2, 3, 4, 5, 6
> coeffs(x^3+2,x) ;
                             2, 1
```
Maple™

coeffs retourne la séquence des coefficients non nuls (seulement) du polynôme, présentés dans l'ordre croissants des puissances de l'indéterminée indiquée. Ce qui dans un exercice d'algèbre linéaire par exemple peut s'avérer inadapté.
Dans ce cas on peut demander explicitement la construction de la séquence des coefficients.

```
> R :=1+p*x+q*x^3 ;
                          R := 1 + px + qx^3
> C :=seq(coeff(R,x,k),k=0..4) ;
                          C := 1, p, 0, q, 0
```
Maple™

N.B : L'instruction **coeff(P,x^0)** retourne un message d'erreur (compréhensible), préférer pour cela toujours la syntaxe **coeff(P,x,k)**.
On peut réordonner l'écriture d'un polynôme suivant les puissances décroissantes d'une indéterminée en utilisant **collect**.

```
> collect(Q,x) ;
```
$$x^{10} - yx^9 - yx^8 + \left(y^2 - y\right)x^7 + \left(y^2 - y\right)x^6$$
$$+2\,y^2x^5 + \left(-y^3 + y^2\right)x^4 - \left(y^2 - y\right)yx^3 - y^3x^2 - y^3x + y^4$$
Maple™

D'autres procédures utiles pour le traitement des expressions polynomiales :

degree(P,x)	pour le degré de P selon l'indéterminée x
expand(P)	pour le developpement de P s'il est donné sous forme factorisée
factor(P)	factoriser P (autant que possible, en fait la factorisation se fait dans Q[X])
factor(P,a)	factoriser P dans l'anneau Q[a][X]
roots(P)	la liste des racines dans le corps $\mathbb{Q}$ de P avec leurs ordres de multiplicité
roots(P,a)	la liste des racines de P se trouvant dans le corps Q[a] avec leurs ordres de multiplicité

```
> P :=(x^2+1)^2*(x-1)^3 ;
                          P := (x^2 + 1)^2 (x - 1)^3
> Q :=expand(P) ;
                  Q := x^7 - 3x^6 + 5x^5 - 7x^4 + 7x^3 - 5x^2 + 3x - 1
> factor(Q) ;
                          (x^2 + 1)^2 (x - 1)^3
> roots(P) ;
                             [[1, 3]]
> roots(P,I) ; # I est le nombre complexe noté usuellement i.
                       [[-I, 2], [I, 2], [1, 3]]
> roots(x^2-2) ;
                               []
> roots(x^2-2,sqrt(2)) ;
                       [[-√2, 1], [√2, 1]]
```
Maple™

II.2 Procédures :

Une procédure maple est ce qui correspondrait à la notion mathématique de fonction.
Pour la construction de fonctions on dispose de l'opérateur **->**(les symboles "moins" et "superieur"), ou de la procédure **unapply**.
Pour la définition de procédures plus avancées, le constructeur de procédure **proc** est le choix ultime.

```
> f :=x->x^2 ;
```

$$f := x \mapsto x^2$$

```
> g :=(x,y)->x*ln(y) ;
```

$$g := (x, y) \mapsto x \ln(y)$$

```
> f(2) ;f(a^2) ;
```

$$4$$
$$a^4$$

```
> g(3,2) ;g(a,b) ;
```

$$3 \ln(2)$$
$$a \ln(b)$$

Maple™

Encore une fois le domaine de définition d'une procédure est strictement formel, et on peut demander la valeur d'une fonction en n'importe quel objet, pourvu que cela ne provoque pas d'erreurs.

```
> A :=matrix(2,2,[1,1,0,2]) ;
```

$$A := \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

```
> f(A) ;
```

$$A^2$$

```
> evalm(f(A)) ;
```

$$\begin{bmatrix} 1 & 3 \\ 0 & 4 \end{bmatrix}$$

Maple™

On expliquera le fait qu'il ait falu passer par evalm pour afficher le résultat dans les prochains paragraphes.
La procédure **unapply** permet de la même façon de construire une fonction à partir d'une expression.

```
> h :=unapply(a*ln(x+1),x) ;
```

$$h := x \mapsto a \ln(x+1)$$

```
> h(b) ;
```

$$a \ln(b+1)$$

Maple™

Et pour une fonction de plusieurs variables :

```
> k :=unapply(a*ln(x+1),x,a) ;
```

$$k := (x, a) \mapsto a \ln(x+1)$$

```
> k(1,3) ;
```

$$3 \ln(2)$$

Maple™

Noter qu'un nom suivi d'un symbole parenthèse fait automatiquement que celui ci est considéré comme le nom d'une procédure, même si ce nom est une constante (il s'agit alors de la fonction constatnte prenant cette valeur).

```
> 1(x) ;1(3) ;
```

$$1$$
$$1$$

```
> g :=h+1 ;
```

$$g := h + 1$$

```
> g(b) ;
```

$$a \ln(b+1) + 1$$

Maple™

Une utilisation intéressante des fonctions, pour le sujet que traite cet article, se fait en association avec **map** :
map(*func, expr*) applique la fonction *func* aux opérandes de *expr*, que *expr* soit une liste, un ensemble ou une expression quelconque.

```
> L :=[k $k=0..5] ;
```

```
> L:=[0,1,2,3,4,5];
```

$$L := [0,1,2,3,4,5]$$

```
> map(k->x^k,L);
```

$$[1, x, x^2, x^3, x^4, x^5]$$

```
> map(k->diff(x^5,x$k),[k $k=1..5]);
```

$$[5x^4, 20x^3, 60x^2, 120x, 120]$$

```
> A:=a+b+c;
```

$$A := a + b + c$$

```
> map(x->x^2,A);
```

$$a^2 + b^2 + c^2$$

```
> f:=x->x^2;
```

$$f := x \mapsto x^2$$

```
> map(f+1,A);
```

$$a^2 + 3 + b^2 + c^2$$

Maple™

map ne fonctionne pas avec les séquences, si vous devez "mapper" la fonction f à une séquence S, faites le plutôt avec la liste $[S]$: **map(f,[S])**

II.3 Résolution de systèmes d'équations algébriques

La procédure **solve** permet de résoudre des équations (ou des systèmes d'équations) algébriques (entre autre), la syntaxe est de la forme

solve(*eqs*,*incs*) ;

où *eqs* est une équation ou un ensemble (au sens Maple) d'équations avec une ou plusieurs indéterminées et *incs* un nom ou un ensemble de noms d' inconnues intervenant dans *eqs*. De cette façon on peut décider que certaine indéterminées jouent le rôle d'inconnues alors que d'autres celui de paramètres. On peut négliger de préciser l'argument *incs*, dans ce cas toutes les indéterminées sont considérées comme des inconnues.

```
> solve(a*x+b=0,x);solve(a*x+b=0);
```

$$-\frac{b}{a}$$

$$\{a = a, b = -ax, x = x\}$$

```
> solve({a*x+b*y=alpha,c*x+d*y=beta},{x,y});
```

$$\left\{x = \frac{d\alpha - \beta b}{ad - cb}, y = \frac{-c\alpha + a\beta}{ad - cb}\right\}$$

Maple™

Maple ne se soucie guère de la validité du résultat dans le cas où des paramètres entent en jeu. Dans le dernier exemple, il a donnée une solution unique du système linéaire que le déterminant $ad - bc$ soit nul ou pas.
Dans le cas où une équation se termine par une égalité avec 0, on peut négliger de préciser =0. Ce détail peut s'avérer très pratique.

```
> solve(a*x+b,x);
```

$$-\frac{b}{a}$$

Maple™

La réponse de Maple est un terme qui représente la solution dans le cas où il y'a une seule inconnue, un ensemble d'égalités dans le cas où il y'en a plusieurs. Bien que dans ce deuxième cas les égalités semblent donner des valeurs aux inconnues, il n'en est rien. Les noms des inconnues restent libres (sans valeurs).

Utilisation de subs pour récupérer les résultats

La syntaxe générale de la procédure **subs** est de la forme

subs(*eqs*,*expr*) ;

où *eqs* est une égalité , une séquence, une liste ou un ensemble d'égalités et *expr* une expression Maple quelconque. Maple va alors remplacer chaque terme de gauche dans le(s) égalité(s) de *eqs* par le terme de droite de la même égalité dans l'expression *expr*. Aucun calcul ne sera effectué, Maple se contentant d'une substitution pure et simple. Noter que la priorité des substitutions sera différente selon est-ce que *eqs* est une liste ou un ensemble.
On peut alors utilser **subs** pour effectivement affecter les valeurs des solutions d'une équation à des noms donnés.

Exemple :

On voudrait déterminer les matrices carrées d'ordre 2 qui commutent avec la matrice

```
> A :=matrix(2,2,[1,1,0,-1]) ;
```

$$A := \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix}$$

Maple™

Pour cela on définit une matrice inconnue X

```
> X :=matrix(2,2,[x,y,z,t]) ;
```

$$X := \begin{bmatrix} x & y \\ z & t \end{bmatrix}$$

Maple™

On évalue la différence $AX - XA$ (voir paragraphe suivant pour les détails de la syntaxe)

```
> M :=evalm(A&*X-X&*A) ;
```

$$M := \begin{bmatrix} z & 2y + t - x \\ -2z & -z \end{bmatrix}$$

Maple™

On récupère l' ensemble des équations et celui des inconnues en utilisant **convert**

```
> eqs :=convert(M,set) ;incs :=convert(X,set) ;
```

$$eqs := \{z, -2z, -z, 2y + t - x\}$$

$$incs := \{t, x, y, z\}$$

Maple™

On résout

```
> sol :=solve(eqs,incs) ;
```

$$sol := \{t = -2y + x, x = x, y = y, z = 0\}$$

Maple™

Maintenant pour voir effectivement la forme des matrices qui commutent avec A

```
> subs(sol,evalm(X)) ;
```

$$\begin{bmatrix} x & y \\ 0 & -2y + x \end{bmatrix}$$

Maple™

III. Initiatin aux boucles et aux instructions conditionnelles.

III.1 Instructions conditionelles avec if

Une instruction conditionnelle sert à exécuter une instruction si une certaine condition est réalisée.

```
if cond1 then
        task1
elif cond2 then
        task2
      ... ...
elif condN then
        taskN
else
        task(N+1)
fi ;
```

va vérifier si cond1 est réalisée, auquel cas task1 sera exécuté et l'instruction se termine, sinon cond2 sera testée, et ainsi de suite, si aucune des condition condK n'est réalise alors l'instruction task(N+1) du bloc **else** sera exécutée et l'instruction se termine. Une instruction conditionnelle n'a d'intérêt que dans le corps d'une procédure.

```
> f :=x->if x<0 then 0 elif x<1 then 1 else 2 fi ;
```

$$f := x \to \text{if } x < 0 \text{ then } 0 \text{ elif } x < 1 \text{ then } 1 \text{ else } 2 \text{ endif}$$

```
> map(f,[-1,1/2,2]) ;
```

$$[0, 1, 2]$$

```
> g :=n->if isprime(n) then n^2 else 0 fi ; # isprime(n) est vraie si n est premier.
```

$$g := n \to \text{if } isprime(n) \text{ then } n^2 \text{ else } 0 \text{ endif}$$

```
> isprime(5) ;isprime(10) ;
```

$$true$$
$$false$$

```
> map(g,[2,4,5,6,9,11]) ;
```

$$[4, 0, 25, 0, 0, 121]$$

Maple™

Pour faire des tests dans une instruction conditionnelle on utilise les opérateurs de comparaisons **<**,**>**,**<=**,**>=** ,**=** et **<>** (pour différent) ; ou des procédures (tel **isprime**) pédéfinies en maple et qui en général donnent une réponse booléenne : true (vrai) ou false (faux). On peut aussi construire ses propres procédures de test. Dans l'exemple suivant on construit la procédure isodd qui vérifie si son argument est impaire ou non.

```
> isodd :=x-> if x mod 2 = 1 then true else false fi ;
```

$$isodd := x \to \text{if } x \bmod 2 = 1 \text{ then } true \text{ else } false \text{ endif}$$

```
> isodd(16) ;isodd(5) ;
```

$$false$$
$$true$$

Maple™

III.2 Boucles for et boucles while :

Une boucle est une instruction adaptée à l'exécution de taches répétitives. On distingue deux types de boucles (et ce dans tous les langages de programmation) :

Les boucles for :

for *k* **from** *a* **to** *b* **by** *p* **do** *task* **od** ;

va exécuter la tâche *task* pour *k* allant de *a* à *b* en avançant d'un pas *p*, si le bloc **by** *p* n'est pas présent le pas par défaut est 1, le bloc **from** *a* peut être aussi absent, auquel cas la boucle commence à partir de 1. Mieux, si le corps *task* ne dépend pas de la valeur de *k*, notre instruction pourrait se limiter à **to** *b* **do** *task* **od** ;, ce qui fera que la tache *task* sera exécutée *b* fois.

Les boucles while :

while *cond* **do** *task* **od** ;

va exécuter la tâche *task* tant que le test de la condition *cond* sera réussi.

Exemples :

```
> a :=2 ;
```

$$a := 2$$

```
> to 5 do a :=2*a od ;
```

$$a := 4$$
$$a := 8$$
$$a := 16$$
$$a := 32$$
$$a := 64$$

Maple™

Ici on initie la variable *a* avec la valeur 2, et on exécute une boucle **for** qui va à chaque fois multiplier la valeur de *a* par 2. Ainsi *a* va recevoir successivement comme valeur les puissances de 2.

```
> L :=[] ;
```

$$L := []$$

```
> for k from 1 to 100 do
> if isprime(k) then L :=[op(L),k] fi
> od ;
> L ;
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]$$

Maple™

Ici la boucle parcourt les entiers de 1 à 100, dés qu'elle tombe sur un nombre premier elle l'ajoute à la liste L qui était au départ vide.
A noter que l'instruction suivante fait la même chose.

```
> select(isprime,[n $n=1..100]) ;
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```
Maple™

La procédure **select** fonctionne selon le même principe que **map**, son premier argument doit être une fonction de test, ce test sera effectué sur les opérandes du deuxième argument. L'instruction ne conservera alors que les opérandes qui vérifient le test.
Un autre exemple (qui parle de lui même) :

```
> select(x->if x>0 then true else false fi , [1,-1,0,2,-5]) ;
[1, 2]
```
Maple™

Un exemple d'utilisation d'une boucle while

```
> a :=160 ;k :=0 ;
                    a := 160
                    k := 0
> while a<>0 mod 2 do k :=k+1 ; a :=iquo(a,2) od ;
                    k := 1
                    a := 80
                    k := 2
                    a := 40
                    k := 3
                    a := 20
                    k := 4
                    a := 10
                    k := 5
                    a := 5
                    k := 6
                    a := 2
                    k := 7
                    a := 1
                    k := 8
                    a := 0
```
Maple™

La boucle vérifie à chaque fois si a ne vaut pas 0 modulo 2, sinon elle remplace a par son quotient par 2 et incrémente k d'une unité. Ce qui permet au final de récupérer dans k l'exposant de 2 dans la décomposition en produit de nombres premiers de la valeur initiale de a :160.

IV. Matrices et vecteurs

IV.1 Différentes méthodes de construction.

Un vecteur est le couple formé de sa taille et de la liste de ses coordonnées, ce qui permet éventuellement de ne pas préciser toutes les coordonnées depuis le départ.

La procédure **vector** permet de construire des vecteurs avec plusieurs variantes dans la syntaxe.
La forme la plus basique, la taille plus la liste des coordonnées :

```
> vector(4,[1,0,0,0]) ;
                    [ 1   0   0   0 ]
```
Maple™

Au lieu de préciser la liste des coefficients, on peut donner une procédure qui servira à construire les coefficients, Maple appliquera alors cette procédure aux indice 1,2,..,taille_vecteur.

```
> vector(10,k->x^(k-1)) ;
        [ 1   x   x^2   x^3   x^4   x^5   x^6   x^7   x^8   x^9 ]
```
Maple™

ou encore

```
> vector(10,1);
```

$$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$$

Maple™

ce deuxième exemple est un cas particulier de la forme précédente : le deuxième argument, ici 1, est traité comme étant la fonction constante de valeur 1. Ceci est pratique pour la construction du vecteur nul.

```
> vector(10,0);
```

$$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

Maple™

Une matrice est la donnée de trois arguments, le nombre de ligne, celui des colonnes et la liste des coefficients lus ligne par ligne.

La procédure **matrix** permet de construire des matrices, et comme pour **vector** avec plusieurs variantes syntaxiques.

```
> matrix(3,3,[1,0,0,0,1,0,0,0,1]);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Maple™

Construit la matrice de taille 3,3 la liste des coefficients est, comme vous pouvez le constater, lue ligne par ligne.

```
> matrix(3,3,(i,j)->1/(i+j-1));
```

$$\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

Maple™

Matrice de Cauchy d'ordre 3, la syntaxe ici ressemble à celle vue pour **vector**.

Pour définir la matrice 3,3 dont les coefficients diagonaux valent a les autres valant b :

```
> A :=matrix(3,3,(i,j)->if i=j then a else b fi);
```

$$A := \begin{bmatrix} a & b & b \\ b & a & b \\ b & b & a \end{bmatrix}$$

Maple™

L' addition de deux matrices (ou de deux vecteurs) se fait en utilisant l'opérateur **+**, leurs multiplication, l'opérateur **&***, si A est une matrice carrée inversible son inverse est simplement **A^(-1)**. On doit toutefois demander explicitement l'évaluation de ces opérations en utilisant **evalm** :

```
> evalm(A&*A);evalm(A^(-1));
```

$$\begin{bmatrix} a^2+2b^2 & 2ab+b^2 & 2ab+b^2 \\ 2ab+b^2 & a^2+2b^2 & 2ab+b^2 \\ 2ab+b^2 & 2ab+b^2 & a^2+2b^2 \end{bmatrix}$$

$$\begin{bmatrix} \frac{a+b}{a^2+ab-2b^2} & -\frac{b}{a^2+ab-2b^2} & -\frac{b}{a^2+ab-2b^2} \\ -\frac{b}{a^2+ab-2b^2} & \frac{a+b}{a^2+ab-2b^2} & -\frac{b}{a^2+ab-2b^2} \\ -\frac{b}{a^2+ab-2b^2} & -\frac{b}{a^2+ab-2b^2} & \frac{a+b}{a^2+ab-2b^2} \end{bmatrix}$$

Maple™

Remarquer ici que pour l'évaluation de l'inverse Maple, ne se soucie pas de l'inversibilté de la matrice si elle contient des paramètres formels et laisse ceci aux soins de l'utilisateur, sauf bien sur, si la matrice ne contient que des coefficients numériques et qu'elle n'est pas inversible, dans ce cas Maple renverra une erreur.

Un comportement intéressant :

```
> evalm(A+1);
```

$$\begin{bmatrix} a+1 & b & b \\ b & a+1 & b \\ b & b & a+1 \end{bmatrix}$$

Maple™

Le 1 dans cette instruction est compris comme étant la matrice scalaire avec 1 sur la diagonale. Il en sera ainsi si on utilise un nom qui n'a pas été défini comme une matrice.

```
> evalm(A-x);
```

$$\begin{bmatrix} a-x & b & b \\ b & a-x & b \\ b & b & a-x \end{bmatrix}$$

Maple™

Evaluation matricielle

L'évaluation d'une expression contenant des noms de matrices ou de vecteurs, s'arrêtera aux simplifications éventuelles sur les noms de ses derniers, il ne seront pas remplacés par leurs valeurs. Par exemple si A est une matrice contenant un coefficient formel *a* l'instruction **subs(a=0,A)** n'aura aucun effet, car Maple se contentera de remplacer *a* par 0 dans le nom "A" de la matrice et non dans son contenu. Pour arriver au résultat escompté ici, il faut demander une substitution explicite dans la valeur de A par **subs(a=1,evalm(A))**.

```
> B :=subs(a=0,A);
```

$$B := A$$

```
> evalm(B);
```

$$\begin{bmatrix} a & b & b \\ b & a & b \\ b & b & a \end{bmatrix}$$

```
> B :=subs(a=0,evalm(A));
```

$$B := \begin{bmatrix} 0 & b & b \\ b & 0 & b \\ b & b & 0 \end{bmatrix}$$

Maple™

Signalons la présence d'autres procédures de construction de matrices de formes particulières (diagonale,symétrique,...), mais qui ne sont pas directement accessible. Il faut auparavant charger le package **linalg** :

```
> with(linalg):
```

Warning, the protected names norm and trace have been redefined and unprotected

```
> diag(1,2,3);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

```
> diag(A,diag(1,1));
```

$$\begin{bmatrix} a & b & b & 0 & 0 \\ b & a & b & 0 & 0 \\ b & b & a & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Maple™

On peut aussi concaténer plusieurs matrices (ou vecteurs) en utilisant **concat**.

```
> M :=k->diag(k,k,k);V :=x->vector(4,[1,x,x^2,x^3]);
```

$$M := k \mapsto \begin{bmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & k \end{bmatrix}$$

$$V := x \mapsto [1, x, x^2, x^3]$$

```
> concat(M(1),M(2),M(3));
```

$$\begin{bmatrix} 1 & 0 & 0 & 2 & 0 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 3 \end{bmatrix}$$

```
> concat(V(a),V(b),V(c),V(d));
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ a & b & c & d \\ a^2 & b^2 & c^2 & d^2 \\ a^3 & b^3 & c^3 & d^3 \end{bmatrix}$$

Maple™

concat s'avère particulièrement interessante pour la construction de matrices de passage.

IV.2 Enfin de l'algebre linéaire :

```
> restart;
```

Maple™

Commençons par charger le package **linalg**.

```
> with(linalg);
```

[BlockDiagonal , GramSchmidt , JordanBlock , LUdecomp , QRdecomp , Wronskian , addcol , addrow , adj , adjoint , angle , augment , backsub , band , basis , bezout , blockmatrix , charmat , charpoly , cholesky , col , coldim , colspace , colspan , companion , concat , cond , copyinto , crossprod , curl , definite , delcols , delrows , det , diag , diverge , dotprod , eigenvals , eigenvalues , eigenvectors , eigenvects , entermatrix , equal , exponential , extend , ffgausselim , fibonacci , forwardsub , frobenius , gausselim , gaussjord , geneqns , genmatrix , grad , hadamard , hermite , hessian , hilbert , htranspose , ihermite , indexfunc , innerprod , intbasis , inverse , ismith , issimilar , iszero , jacobian , jordan , kernel , laplacian , leastsqrs , linsolve , matadd , matrix , minor , minpoly , mulcol , mulrow , multiply , norm , normalize , nullspace , orthog , permanent , pivot , potential , randmatrix , randvector , rank , ratform , row , rowdim ,

rowspace , rowspan , rref , scalarmul , singularvals , smith , stackmatrix , submatrix , subvector , sumbasis , swapcol , swaprow , sylvester , toeplitz , trace , transpose , vandermonde , vecpotent , vectdim , vector , wronskian]

Remarquer la liste des procédures maintenant disponibles, et dont certaines portent des noms évocateurs. Voici dans leur ordre d'apparition, une courte description de celle qui sont les plus utiles pour nous. A étant une matrice donnée (une matrice carrée si nécessaire) et U et V deux *listes* de vecteurs qui ont la même taille.

basis(V) retourne une famille libre maximale extraite de la famille de vecteurs V : une base de Vect(V).

charpoly(A,x) retourne le polynôme caractéristique de A, x étant le nom de l'indéterminée que l'on veut utiliser.

cholesky(A) si A est une matrice symétrique positive, retourne une matrice triangulaire supérieure T telle que $A = {}^tTT$

col(A,k) retourne la $k^{ème}$ colonne de A.

coldim(A) retourne le nombre de colonne de A. Utile si on veut écrire une procédure avancée faisant intervenir des matrices. **rowdim(A)** donne le nombre de lignes de A.

colspace(A) retourne une base du sev engendré par les vecteurs colonnes de A.

concat à déjà été décrite.

delcols(A,k..h) permet d'éliminer les colonnes d'indices de k à h de A.

delrows(A,k..h) la même chose pour les lignes.

det(A) n'a pas besoin de description !

eigenvals(A) retourne la séquence des valeurs propres de A, chacune étant répétée autant de fois que sa multiplicité.

eigenvects(A) retourne un résultat composite comprenant les valeurs propres de A, leurs multiplicités et une base de chaque sous espace propre. Suffisant pour voir si une matrice est diagonalisable ou pas.

equal(A,B) retourne true (vrai) si les matrices A et B sont égales , false (faux) sinon. Utile dans une instruction conditionnelles ou dans une boucle while.

gausselim(A) retourne la réduite de la matrice A par la méthode de Gauss.

intbasis(U,V) retourne une base de l'intersection des sevs engendrés par les familles de vecteurs U et V.

inverse(A) retourne l'inverse de A si c'est possible, un message d'erreur sinon. on peut obtenir la même chose avec **evalm(A^(-1))**.

jordan(A) retourne la réduite de Jordan de la matrice A, en fait une matrice diagonale ou triangulaire supérieure (quand c'est possible) semblable à A.

jordan(A,'P') retourne la réduite de Jordan de A et stocke la matrice de passage dans le nom P, ainsi on peut réduire une matrice et récupérer la matrice de passage par une seule instruction.

kernel(A) retourne une base du noyau de A. **nullspace(A)** fait la même chose.

linsolve(A,B) donne les solutions de l'équation AX=B, A et B étant des matrices qui ont le même nombre de lignes. les solutions sont exprimées à l'aide de paramètres internes de la forme _t[1], _t[2],.. on peut leur donner des valeurs particulières en utilisant **subs**.

minpoly(A,x) retourne le polynôme minimal de A.

QRdecomp(A,Q='P') retourne une matrice triangulaire supérieure R et stocke dans le nom P une matrice orthogonale telles que $A = PR$.

randmatrix(n,m) construit une matrice de taille n,m avec des coefficients aléatoires.

rank(A) le rang de A.

rowspace(A) une base du sev engendré par les vecteurs lignes de A.

sumbasis(U,V) retourne une base de la somme des sevs engendrés par les familles de vecteurs U et V.

trace(A) la trace de A.

transpose(A) la transposée de A.

Exemple 1 : Réduction de matrices

```
> A :=randmatrix(4,4) ;
```

$$A := \begin{bmatrix} -7 & 22 & -55 & -94 \\ 87 & -56 & 0 & -62 \\ 97 & -73 & -4 & -83 \\ -10 & 62 & -82 & 80 \end{bmatrix}$$

```
> B :=matrix(4,4,1) ;
```

$$B := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Maple™

```
> charpoly(A,x) ;
```

$$11543578 - 662225\,x - 5197\,x^2 - 13\,x^3 + x^4$$

```
> factor(charpoly(A,x)) ; #problèmes en vue.
```

$$11543578 - 662225\,x - 5197\,x^2 - 13\,x^3 + x^4$$

```
> eigenvals(A);
```

$$RootOf\left(11543578 - 662225\,\_Z - 5197\,\_Z^2 - 13\,\_Z^3 + \_Z^4, index = 1\right),$$
$$RootOf\left(11543578 - 662225\,\_Z - 5197\,\_Z^2 - 13\,\_Z^3 + \_Z^4, index = 2\right),$$
$$RootOf\left(11543578 - 662225\,\_Z - 5197\,\_Z^2 - 13\,\_Z^3 + \_Z^4, index = 3\right),$$
$$RootOf\left(11543578 - 662225\,\_Z - 5197\,\_Z^2 - 13\,\_Z^3 + \_Z^4, index = 4\right)$$

Maple™

$RootOf(P(\_Z), index = k)$ signifie racine de P, indexée par k pour différencier entre les racines. Malin Maple ! On peut quand même forcer l'évaluation en utilisant **allvalues**.

```
> map(allvalues,[eigenvals(A)]) : #très couteux en photocopies donc passé sous silence.
```

Maple™

Changeons pour une matrice plus gentille.

```
> eigenvals(B);
```

$$0, 0, 0, 4$$

```
> epropre :=eigenvects(B);
```

$$epropre := [0, 3, \{[\ -1\ \ 0\ \ 0\ \ 1\ ], [\ -1\ \ 0\ \ 1\ \ 0\ ], [\ -1\ \ 1\ \ 0\ \ 0\ ]\}],$$
$$[4, 1, \{[\ 1\ \ 1\ \ 1\ \ 1\ ]\}]$$

Maple™

4 est une va.p de multiplicité 1 et un vecteur dans la base du sous espace propre, 0 est une valeur propre de multiplicité 3 et 3 vecteurs dans la base du sep. La matrice B est diagonalisable. On peut récupérer la base de vecteurs propres par :

```
> vpropre :=map(L->op(L[3]),[epropre]);
```

$$vpropre := [[\ -1\ \ 1\ \ 0\ \ 0\ ], [\ -1\ \ 0\ \ 1\ \ 0\ ], [\ -1\ \ 0\ \ 0\ \ 1\ ],$$
$$[\ 1\ \ 1\ \ 1\ \ 1\ ]]$$

Maple™

epropre est une séquence dont les opérandes sont des *listes* de la forme $[valpropre, mult, \{base\ du\ sep\}]$, on 'mappe' alors à la liste $[epropre]$, la fonction qui à une liste L associe la séquence des opérandes de son troisième élément, ici la séquence des vecteurs de la base du sous espace propre. Rappelez-vous que 'mapper' à la séquence epropre ne marchera pas. Le résultat est une liste constituée des vecteurs d'une base de vecteur propre.
Former ensuite la matrice de passage par :

```
> P :=concat(op(vpropre));
```

$$P := \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Maple™

et vérifier la formule de passage :

```
> equal(B,P&*diag(0,0,0,4)&*P^(-1));
```

$$true$$

Maple™

On peut aussi déterminer les vecteurs propres à la main :

```
> vec1 :=linsolve(B-4,vector(4,0));
```

$$vec1 := [\ \_t_1\ \ \_t_1\ \ \_t_1\ \ \_t_1\ ]$$

Maple™

On récupère une base pour chaque sous espace propre par :

```
> e1 :=subs(_t[1]=1,evalm(vec1));
> vec2 :=linsolve(B,vector(4,0));
> e2 :=subs(_t[1]=1,_t[2]=0,_t[3]=0,evalm(vec2));
> e3 :=subs(_t[1]=0,_t[2]=1,_t[3]=0,evalm(vec2));
> e4 :=subs(_t[1]=0,_t[2]=0,_t[3]=1,evalm(vec2));
```

$$e1 := \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$vec2 := \begin{bmatrix} -\_t_1 - \_t_2 - \_t_3 & \_t_1 & \_t_2 & \_t_3 \end{bmatrix}$$

$$e2 := \begin{bmatrix} -1 & 1 & 0 & 0 \end{bmatrix}$$

$$e3 := \begin{bmatrix} -1 & 0 & 1 & 0 \end{bmatrix}$$

$$e4 := \begin{bmatrix} -1 & 0 & 0 & 1 \end{bmatrix}$$

Maple™

La matrice de passage maintenant :

```
> concat(e1,e2,e3,e4);
```

$$\begin{bmatrix} 1 & -1 & -1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Maple™

Exemple 2 : Un endomorphisme de $\mathbb{R}_4[X]$

On considère l'endomorphisme T de $R_4[X]$ défini par

$$T(P) = 2XP' + (1 - X^2)P''$$

Écriture de la matrice de T dans la base canonique :

On construit d'abords une procédure qui à un élément de $R_4[X]$ retourne le vecteur colonne de ses coordonnées dans la base canonique de $R_4[X]$.

```
> coord :=P->vector(5,[seq(coeff(P,X,k),k=0..4)]);
```

$$coord := P \mapsto vector(5, [seq(coeff(P, X, k), k = 0..4)])$$

Maple™

On définit la procédure T et on récupère les vecteurs des coordonnées des polynômes $T(X^k)$.

```
> T :=P->expand(2*X*diff(P,X)+(1-X^2)*diff(P,X,X));
> T(X^4);
> coord(T(X^4));
> vects :=map(coord,[T(X^k) $k=0..4]);
```

$$T := P \mapsto expand(2 * X * (diff(P, X)) + (1 - X^2) * (diff(P, X, X)))$$

$$-4\,X^4 + 12\,X^2$$

$$\begin{bmatrix} 0 & 0 & 12 & 0 & -4 \end{bmatrix}$$

$$vects := [[\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 2 & 0 & 2 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 6 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 12 & 0 & -4 \end{bmatrix}]]$$

Maple™

La matrice de T dans la base canonique maintenant :

```
> M :=concat(op(vects));
> eigenvects(M);
```

$$M := \begin{bmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 6 & 0 \\ 0 & 0 & 2 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -4 \end{bmatrix}$$

$$[0, 2, \{[\,1\ \ 0\ \ 0\ \ 0\ \ 0\,], [\,0\ \ -3\ \ 0\ \ 1\ \ 0\,]\}],$$
$$[-4, 1, \{[\,1\ \ 0\ \ -2\ \ 0\ \ 1\,]\}],$$
$$[2, 2, \{[\,0\ \ 1\ \ 0\ \ 0\ \ 0\,], [\,1\ \ 0\ \ 1\ \ 0\ \ 0\,]\}]$$

Maple™

Où on voit que T est diagonalisable de valeurs propres $(0, 2)$, $(2, 2)$ et $(-4, 1)$.
Pour récupérer les vecteurs propres on pourrait écrire la procédure inverse de **coord** :

```
> icoord :=V->sum(V[k]*X^(k-1),k=1..5);
```

$$icoord := V \mapsto V_1 + V_2 X + V_3 X^2 + V_4 X^3 + V_5 X^4$$

```
> v :=vector(5,[0,1,1,0,1]);
```

$$v := [\,0\ \ 1\ \ 1\ \ 0\ \ 1\,]$$

```
> icoord(v);
```

$$X + X^2 + X^4$$

```
> VECTpropre :=map(L->op(L[3]),[eigenvects(M)]);
```

$$VECTpropre := [[\,0\ \ -3\ \ 0\ \ 1\ \ 0\,], [\,1\ \ 0\ \ 0\ \ 0\ \ 0\,],$$
$$[\,1\ \ 0\ \ -2\ \ 0\ \ 1\,], [\,1\ \ 0\ \ 1\ \ 0\ \ 0\,], [\,0\ \ 1\ \ 0\ \ 0\ \ 0\,]]$$

```
> POLpropre :=map(icoord,VECTpropre);
```

$$POLpropre := [-3X + X^3, 1, 1 - 2X^2 + X^4, 1 + X^2, X]$$

Maple™

Exemple 3 : Recherche d'une matrice à coefficients entiers et de déterminant 1.

On définit d'abords une procédure qui aura pour rôle de donner au hasard un entier compris entre -2 et 2 :

```
> haz :=rand(-2..2);
```

$$haz := \text{proc}()\ \text{proc}()\ \text{option}\ builtin;\ 391\ \text{endproc}\,(6, 5, 3) - 2\ \text{endproc}$$

Maple™

haz est maintenant une procédure qui ne prend aucun argument et qui génère à chaque appel un entier entre -2 et 2.

```
> haz();
```

$$1$$

```
> haz();
```

$$-2$$

Maple™

Une matrice au hazard à coefficients entiers entre -2 et 2 :

```
> M :=matrix(4,4,haz);
```

$$M := \begin{bmatrix} 2 & 0 & -1 & -2 \\ -1 & 1 & 2 & 0 \\ 1 & -2 & -2 & -2 \\ -1 & -1 & 0 & 1 \end{bmatrix}$$

Maple™

Maintenant une boucle **while** qui ne s'arrêtera que lorsque elle trouve une matrice de det 1 :

```
> while det(M) <> 1 do M :=matrix(4,4,haz) od :
> evalm(M) ;det(M) ;
```

$$\begin{bmatrix} 0 & 1 & 1 & -2 \\ -2 & 0 & 1 & 1 \\ 2 & 1 & -1 & -1 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$
$$1$$

Maple™

Une telle matrice peut servir pour construire une matrice diagonalisable (ou trigonalisable) qui conserve des coefficients entiers et qui aura des valeurs propres décidées à l'avance (truc de profs de prépas) :

```
> A :=evalm(M&*diag(0,1,1,-1)&*M^(-1)) ;
```

$$A := \begin{bmatrix} -3 & 8 & 4 & -8 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 2 \\ 1 & -2 & -1 & 2 \end{bmatrix}$$

Maple™

Exemple 4 : Décomposition de Dunford en utilisant (en trichant) Jordan

On considère la matrice suivante (fabriquée par le procédé de l'exemple 3)

```
> A :=matrix([[-1,2,2,4,0],[-4,20,15,16,2],[7,-35,-26,-27,-4],[-3,14,10,10,2],[5,-25,-17,-16,-3]]) ;
```

$$A := \begin{bmatrix} -1 & 2 & 2 & 4 & 0 \\ -4 & 20 & 15 & 16 & 2 \\ 7 & -35 & -26 & -27 & -4 \\ -3 & 14 & 10 & 10 & 2 \\ 5 & -25 & -17 & -16 & -3 \end{bmatrix}$$

Maple™

"Jordanisation"

```
> T :=jordan(A,'P') ;
```

$$T := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> evalm(P) ;
```

$$\begin{bmatrix} -6 & 2 & \frac{68}{9} & 0 & 3/2 \\ -3 & 1 & \frac{16}{9} & 3/2 & 1/4 \\ 6 & -1 & -\frac{7}{9} & -3/2 & -7/4 \\ -3 & 0 & 0 & 0 & 3/2 \\ -3 & 1 & \frac{25}{9} & -3 & 5/2 \end{bmatrix}$$

Maple™

Une petite vérification :

```
> equal(P&*T&*P^(-1),A) ;
```

true

Maple™

Le petit malin remarquera que le travail de diagonalisation fait dans l'exemple 1 est de ce fait obsolète !
On récupère la matrice diagonale L formée des éléments diagonaux de T et la matrice nilpotente $N = T - L$.

```
> L :=diag(T[k,k] $k=1..5) ; N :=evalm(T-L) ;
```

$$L := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$N := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Maple™

Retour à la base canonique :

```
> K :=evalm(P&*L&*P^(-1)) ;M :=evalm(P&*N&*P^(-1)) ;
```

$$K := \begin{bmatrix} -1 & 0 & 0 & 2 & 0 \\ -1 & 5 & 4 & 5 & 0 \\ 4 & -20 & -15 & -16 & -2 \\ -3 & 14 & 10 & 10 & 2 \\ -1 & 2 & 2 & 3 & 1 \end{bmatrix}$$

$$M := \begin{bmatrix} 0 & 2 & 2 & 2 & 0 \\ -3 & 15 & 11 & 11 & 2 \\ 3 & -15 & -11 & -11 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 6 & -27 & -19 & -19 & -4 \end{bmatrix}$$

Maple™

Les vérifications maintenant

```
> equal(A,K+M) ;equal(K&*M,M&*K) ;equal(M^5,matrix(5,5,0)) ;
```

true
true
true

Maple™

Exemple 5

Il s'agit ici de voir à quelle condition la matrice suivante, A, est-elle diaginalisable ?

```
> A :=matrix(5,5,[a,0,0,0,b,0,a,0,b,0,0,1,2,1,0,0,b,0,a,0,b,0,0,0,a]) ;
```

$$A := \begin{bmatrix} a & 0 & 0 & 0 & b \\ 0 & a & 0 & b & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & b & 0 & a & 0 \\ b & 0 & 0 & 0 & a \end{bmatrix}$$

Maple™

Les éléments propres de A.

```
> EP :=eigenvects(A) ;
```

$$EP := [2, 1, \{[\,0\ \ 0\ \ 1\ \ 0\ \ 0\,]\}],$$
$$[b+a, 2, \{[\,1\ \ 0\ \ 0\ \ 0\ \ 1\,], [\,0\ \ 1/2\,b+1/2\,a-1\ \ 1\ \ 1/2\,b+1/2\,a-1\ \ 0\,]\}]$$
$$[-b+a, 2, \{[\,0\ \ -1\ \ 0\ \ 1\ \ 0\,], [\,-1\ \ 0\ \ 0\ \ 0\ \ 1\,]\}]$$

Maple™

En apparence donc A est diagonalisable. Mais voilà, les valeurs propres 2, $a - b$, $a + b$ ne sont pas forcément deux à deux distinctes pour ceraines valeurs de a et b (Maple s'en soucie peu) , et dans ce cas la famille de vecteurs propres calculée va-t-elle toujours constituer une base ? Pour en être sûr nous allons constituer la matrice de passage dans cette famille (de vecteurs propres) et vérifier à la main dans quel cas elle va être non inversible. On crée ensuite une matrice B qui correspond à A dans ce cas là.

La séquence des vecteurs propres de A :

```
> VP :=seq(op(EP[k][3]),k=1..3) ;
```

$$VP := [\,0\ \ 0\ \ 1\ \ 0\ \ 0\,], [\,1\ \ 0\ \ 0\ \ 0\ \ 1\,],$$
$$[\,0\ \ 1/2\,b+1/2\,a-1\ \ 1\ \ 1/2\,b+1/2\,a-1\ \ 0\,],$$
$$[\,0\ \ -1\ \ 0\ \ 1\ \ 0\,], [\,-1\ \ 0\ \ 0\ \ 0\ \ 1\,]$$

Maple™

La matrice de passage :

```
> P :=concat(VP) ;
```

$$P := \begin{bmatrix} 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1/2\,b + 1/2\,a - 1 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1/2\,b + 1/2\,a - 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Maple™

Dèterminant de P pour voir dans quels cas elle est non inversible : a+b=2

```
> det(P) ;
```

$$2\,b + 2\,a - 4$$

```
> CAS :=solve(det(P)) ;
```

$$CAS := \{a = -b + 2, b = b\}$$

Maple™

On construit la matrice B qui correspond à A dans le cas où a+b=2.

```
> B :=subs(CAS,evalm(A)) ;
```

$$B := \begin{bmatrix} -b+2 & 0 & 0 & 0 & b \\ 0 & -b+2 & 0 & b & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & b & 0 & -b+2 & 0 \\ b & 0 & 0 & 0 & -b+2 \end{bmatrix}$$

```
> eigenvects(B) ;
```

$$[2, 3, \{[\,1 \quad 0 \quad 0 \quad 0 \quad 1\,], [\,0 \quad 0 \quad 1 \quad 0 \quad 0\,]\}],$$
$$[-2\,b + 2, 2, \{[\,0 \quad -1 \quad 0 \quad 1 \quad 0\,], [\,-1 \quad 0 \quad 0 \quad 0 \quad 1\,]\}]$$

Maple™

On voit alors que dans le cas où a+b=2, la matrice A est non diagonalisable.

❑