CHAPITRE

3

Autour du calcul matriciel

1 Étude de quelques algorithmes

1 1 Fabriquons nos outils

1 1 a Les primitives

Les matrices seront crées comme une liste de listes représentant les lignes.

Par exemple, $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$ sera creée avec :

```
> m := [[1,2],[3,4],[5,6]]:
```

On obtiendra alors $m_{i,j}$ en entrant `m[i][j]`.
Pour avoir un affichage convivial, on crée une procédure convertissant la liste en tableau (type `array`) :

```
> affiche:=proc(T)
    print(convert(T,array))
  end:
```

On obtient l'affichage de le matrice en entrant `affiche(m)`.
On aura également besoin de deux primitives `Lignes` et `Cols` qui « compterons » le nombre de lignes et de colonnes de la matrice :

```
> Lignes := proc(M)
    RETURN(nops(M))
  end:

> Cols := proc(M)
    RETURN(nops(M[1]))
  end:
```

On aura souvent besoin de créer une matrice nulle d'une certaine taille :

```
> nulle := proc(lignes,colonnes)
    RETURN([[0 $ lignes] $ colonnes])
  end:
```

1 1 b Transposée d'une matrice

Déterminez une procédure `transpose := proc(M)` qui renvoie la transposée d'une matrice M.

1 1 c Unité

Déterminer une procédure `Id := proc(n)` qui renvoie la matrice unité d'ordre n.

1 2 Pivot de Gauss

1 2 a Opérations élémentaires

Créer tout d'abord une procédure `GJ(M) := proc(M)` qui fabrique une matrice où se juxtaposent la matrice initiale et la matrice identité.

Par exemple, $M = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ est complétée en :

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

On montre ensuite qu'en effectuant des combinaisons sur les lignes ou des permutations de lignes, si on arrive à obtenir la matrice unité dans la partie gauche du tableau alors M est inversible et son inverse est la partie droite du tableau.
On commence par créer une procédure `mult_ligne := proc(k,M,j)` qui effectue la transformation $L_j \leftarrow k \times L_j$.

```
> mult_ligne := proc(k,M,i)
  local A,j;
  A := M;
  for j from 1 to Cols(M) do
    A[i][j] := k*M[i][j]
  od:
  RETURN(A)
  end:
```

Selon le même modèle, créer une procédure `comb_lignes := proc(ki,kj,M,i,j)` qui effectue la transformation $L_j \leftarrow k_i \times L_i + k_j \times L_j$.
Créer aussi une procédure `swap := proc(M,i,j)` qui effectue l'échange de lignes $L_i \leftrightarrow L_j$.
On peut alors effectuer une succession d'opérations élémentaires pour vérifier si M et inversible et obtenir alors son inverse :

```
> T := GJ(M): affiche(T);
```

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

```
> T := comb_lignes(-1,1,T,1,3): affiche(T);
```

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

On continue ainsi jusqu'à obtenir :

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

Il reste à extraire la partie gauche du tableau en créant une procédure `JG := proc(T)`. On se souviendra que `L[k..m]` renvoie la sous-liste de L contenant les éléments de `L[k]` jusque `L[m]`.

1 2 b Calcul direct de l'inverse par la méthode de Gauss-Jordan

lLidée est de balayer le tableau par colonne.
Étant donné une colonne, on cherche un élément non nul. S'il n'y en a pas, la matrice n'est pas inversible et le système n'admet pas une unique solution ; sinon, on permute éventuellement deux lignes pour placer l'élément non nul de la colonne k sur la ligne k et on divise tous les éléments de la ligne par le nouveau a_{kk} pour obtenir 1.
Il reste ensuite à remplacer chaque ligne (autre que L_k) dont l'élément de la colonne k est non nul par $L_i - a_{ik} \times L_k$.
Analyser le programme suivant :

```
> inv_gauss := proc(M)
    local S,NoLigne,NoCol,E,k,pivot;
    if Cols(M) <>  Lignes(M) then
       RETURN('La matrice doit être carrée')
    fi;
    S := GJ(M);
    for NoCol from 1 to Lignes(M) do
       NoLigne := NoCol;
       while (S[NoLigne][NoCol] = 0) do
         NoLigne := NoLigne + 1;
         if (NoLigne = Lignes(M)) then
            RETURN('Matrice non inversible')
         fi;
       od;
       S := swap(S,NoLigne,NoCol);
       pivot := S[NoLigne][NoCol];
       S := mult_ligne(1/pivot,S,NoCol);
       E := {seq(k,k=1..Lignes(T))} minus {NoCol};
       for k in E do
          S := comb_lignes(-S[k][NoCol],1,S,NoCol,k);
       od;
     od;
     RETURN(JG(S))
  end:
```

1 3 Généralisation

Afin de résoudre des systèmes linéaires, il ne faut pas se limiter aux matrices inversibles ni même aux matrices carrées.
On va essayer d'écrire la matrice originale sous forme « triangulaire supérieure » en généralisant cette configuration aux matrices « rectangulaires ».
On s'inspire de la fonction précédente, mais au lieu de travailler sur toutes les lignes, on ne transforme que les lignes en-dessous du pivot.
Il faudra donc faire attention maintenant à utiliser le minimum entre le nombre de lignes et le nombre de colonnes de la matrice.

```
> A:=[[4,1,-2,8,1],[5,-13,7,6,1],[-1,9,-11,9,1],[2,-8,12,-3,1]]:
> affiche(A);
```

$$\begin{bmatrix} 4 & 1 & -2 & 8 & 1 \\ 5 & -13 & 7 & 6 & 1 \\ -1 & 9 & -11 & 9 & 1 \\ 2 & -8 & 12 & -3 & 1 \end{bmatrix}$$

```
> affiche(tri_gauss(A));
```

$$\begin{bmatrix} 4 & 1 & -2 & 8 & 1 \\ 0 & -57 & 38 & -16 & -1 \\ 0 & 0 & 1216 & -1916 & -248 \\ 0 & 0 & 0 & -1924320 & -594624 \end{bmatrix}$$

1 3 a Base de l'image - Rang d'une matrice

On écrit les vecteurs engendrant l'image de l'endomorphisme en ligne : bref, on transpose la matrice. En effet, ce sera plus simple d'étudier la matrice par lignes car il s'agit des sous-listes de la liste représentant la matrice. Suite à la réduction de `tri_gauss` on ne garde que les lignes non nulles : la famille obtenue est une base de l'image.
Déterminer une fonction `image := proc(M)` qui retourne une base de l'image puis une fonction `rang := proc(M)` qui renvoie le rang de la matrice M.

1 3 b Calcul du déterminant

On utilise `tri_gauss` sur une matrice carrée : on obtient ainsi une matrice triangulaire. Le déterminant est égal au produit des éléments diagonaux à un détail près : il ne faut pas oublier de tenir compte de la parité du nombre d'échanges de lignes ainsi que des multiplications des lignes modifiées par combinaisons.

1 3 c Polynôme caractéristique et valeurs propres

Déduisez-en une procédure `polycar := proc(M)` qui renvoie le polynôme caractéristique de M.
On pourra introduire `idx := proc(n,x)`, une variante de `id(n)`, qui renvoie $x\mathrm{I}_n$, puis une procédure `Ex := proc(M,x)` qui renvoie la matrice $\mathrm{M} - x\mathrm{I}_n$.

```
> B := [[1,1,0],[-1,-1,0],[1,-1,0]]:
> affiche(Ex(B,x))
```

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

```
> polycar(B);
```

$$-x^3$$

Comment déterminer à l'aide de Maple si cette matrice est diagonalisable ?
Déterminer un test `est_diagonalisable := proc(M)` qui renvoie vrai ou faux.

2 Maple à Centrale

3 - 1 Centrale, 2010

Si $n \geq 2$, on note $= (\mathbf{e}_1, ..., \mathbf{e}_n)$ la base canonique de $\mathbb{R}^n$.
Soit $f_n \in \mathcal{L}(\mathbb{R}^n)$ tel que $f_n(\mathbf{e}_i) = \mathbf{e}_{i+1}$ si $i \in [\![1, n-1]\!]$ et $f_n(\mathbf{e}_n) = \mathbf{e}_1$.

1. Écrire une procédure permettant d'obtenir la matrice A_n de f_n dans la base $\mathbf{e}$.
2. Dans cette question, $n = 6$.
   (a) Calculer le polynôme caractéristique de A_6. Factoriser ce polynôme sur $\mathbb{R}[X]$.
   On écrit $\chi = \prod_{1 \leq k \leq p} R_k$ où les R_k sont irréductibles dans $\mathbb{R}[X]$.
   (b) Déterminer $\ker R_k(f_6)$ pour $k \in [\![1, p]\!]$. Montrer que $\mathbb{R}^6 = \ker R_1(f_6) \oplus \cdots \oplus \ker R_p(f_6)$.
   (c) Donner la matrice de f_6 dans une base adaptée à cette décomposition.
3. Si $n \geqslant 2$, calculer le polynôme caractéristique χ de A_n. Le factoriser sur $\mathbb{R}[X]$.

Voyons quelques commandes utiles issues de la bibliothèque `LinearAlgebra` :

```
with(LinearAlgebra);
# a)
A := proc(n);
    M := Matrix(n);
    for i to n-1 do
        M[i+1, i] := 1;
        end do;
    M[1, n] := 1;
    return M
    end proc;

M := A(6);
I6 := IdentityMatrix(6);
latex(M);

# b)
#    i)
chi := CharacteristicPolynomial(M, x);
chi := factor(chi);
R := op(chi);

#    ii)
# Détermination des sev Ker(Rk(f))
F1 := NullSpace(M - I6);
F2 := NullSpace(M + I6);
F3 := NullSpace(M^2 + M + I6);
F4 := NullSpace(M^2 - M + I6);

P := <F1[1]|F2[1]|F3[1]|F3[2]|F4[1]|F4[2]>;
# Trois manières de vérifier que P est inversible :
Determinant(P);
NullSpace(P);
ColumnSpace(P);
```

```
#    iii)
# Matrice de f_6 dans un base adaptée :
P^(-1).M.P;

# c)
for n from 2 to 10 do
    CharacteristicPolynomial(A(n), x),\
    factor(CharacteristicPolynomial(A(n), x));
    end do;
```

3 - 2 Centrale, 2010

1. Démontrer que, si deux endomorphismes u et v d'un espace vectoriel E commutent, alors, les sous-espaces propres de u et l'image de u sont stables par v.
   Dans les deux cas suivants :

$$A = \begin{pmatrix} 20 & 12 & -4 & 12 \\ -4 & -3 & 9 & -5 \\ -4 & 1 & 5 & -5 \\ -8 & -10 & 6 & -2 \end{pmatrix} \quad \text{et} \quad A = \begin{pmatrix} -12 & -16 & -8 & -4 \\ 4 & 13 & 1 & -1 \\ 4 & 5 & 9 & -1 \\ 8 & 10 & 2 & 6 \end{pmatrix}$$

2. Préciser les matrices qui commutent avec A (structure, dimension, base éventuelle).
3. Etudier dans $\mathcal{M}_4(\mathbb{R})$, puis dans $\mathcal{M}_4(\mathbb{C})$, l'équation

$$X^2 = A$$

   (nombre de solutions, un exemple de solution quand il y en a, somme et produit des solutions quand elles sont en nombre fini).

```
with(LinearAlgebra):
# a)
A := <<20|12|-4|12>, <-4|-3|9|-5>, <-4|1|5|-5>, <-8|-10|6|-2>>;
(vpA, P) := Eigenvectors(A);
P^(-1).A.P;

B := <<-12|-16|-8|-4>, <4|13|1|-1>, <4|5|9|-1>, <8|10|2|6>>;
(vpB, Q) := Eigenvectors(B);
Q^(-1).B.Q;

# c)
Determinant(A);
Determinant(B);
```