ALGORITHMIQUE ALGEBRIQUE 2

Alain Yger

Table des matières

CHAPITRE 1

Matrices et applications linéaires

1.1. Matrice d'une application linéaire, manipulations de base sous Sage

Dans cette section, $\mathbb{K}$ désigne un corps commutatif (fini ou infini), `M` et `N` deux entiers strictement positifs. Une *matrice* à `M` lignes et `N` colonnes à entrées dans $\mathbb{K}$ est par définition un *tableau* `A` à `M` lignes et `N` colonnes dont les entrées sont des éléments du corps $\mathbb{K}$. L'entrée correspondant à la ligne d'indice $j \in \{0, ..., \mathtt{M}-1\}$ et à la colonne d'indice $k \in \{0, ..., \mathtt{N}-1\}$ est notée $a_{j,k}$: le premier indice sera donc dans tout ce cours l'indice de ligne, tandis que le second indice figurera l'indice de colonne.

Sous le logiciel `Sage` (qui illustrera d'un bout à l'autre ce cours), voici comment se déclare une telle matrice :

```
sage :
A = Matrix(corps, [Liste_0, Liste_2,...,Liste_(M-1)])
```

Ici les `M` listes `Liste_`j, $j = 0, ..., \mathtt{M}-1$ sont des listes $[a_{j,0}, ..., a_{j,\mathtt{N}-1}]$ composées chacune de `N` entrées toutes déclarées dans le corps $\mathbb{K}$ (ici `corps`).

Voici quelques manipulations simples sur les matrices utilisant sous le logiciel `Sage` : la première isole une entrée spécifique, indexée par `j` entre `0` et `M-1` (`M` figurant le nombre de lignes, on rappelle que l'indexation d'une liste de `M` objets sous `Python` se fait suivant la numérotation `0,1,....,M-1` [1]) et un indice `k` entre `0` et `N-1` (`N` figurant ici le nombre de colonnes de `A`) ; la seconde génère une *matrice extraite* correspondant à une liste de lignes et une liste de colonnes prescrites.

```
sage:
A = Matrix(QQ,[[-1/2,8,5,4],[3,1,-1/7,5],
          [4,1,0,-3/5],[0,1,0,4],[-1,1,2/5,0]]); A
reponse:
[-1/2    8    5    4]
[   3    1 -1/7    5]
[   4    1    0 -3/5]
[   0    1    0    4]
[  -1    1  2/5    0]
sage:
A[3,2], A[0,3]
reponse:
(0,4)
```

1. C'est ce qui justifie nos notations dans ce cours.

```
sage:
A.matrix_from_rows_and_columns([0,1],[2,3])
reponse:
[   5    4]
[-1/7    5]
```

La routine `A.transpose()` transforme d'autre part une matrice de taille [`M`, `N`] en sa transposée : les nombres de lignes et de colonnes (respectivement `A.nrows()` et `A.ncols()`) sont échangés et l'entrée $a_{j,k}$ est transformée en l'entrée $a_{k,j}$; cette matrice transposée devient donc de taille [`N`, `M`]. Si enfin `A` est une matrice à `M =A.nrows()` lignes et `N = A.ncols()` colonnes et `j` un indice entre `0` et `M-1`, `A[j]` figure un `vecteur` sous `Sage`, élément de $\mathbb{K}^{\mathtt{N}}$ dont la liste des `N` coordonnées est donnée par `A[j].list()` ; il s'agit du vecteur ligne d'indice j de la matrice `A` .

```
sage:
A = Matrix(QQ,[[-1/2,8,5,4],[3,1,-1/7,5],
          [4,1,0,-3/5],[0,1,0,4],[-1,1,2/5,0]]); A
reponse:
[-1/2    8    5    4]
[   3    1 -1/7    5]
[   4    1    0 -3/5]
[   0    1    0    4]
[  -1    1  2/5    0]

sage: A[3]; type(A[3]); A[3].list()
reponse:
(0, 1, 0, 4)
<type 'sage.modules.vector_rational_dense.
                    Vector_rational_dense'>
[0, 1, 0, 4]
```

On pourra envisager comme corps $\mathbb{K}$ un corps relevant du calcul exact (sans pertes). Les exemples type relevant de ce cadre sont ceux de $\mathbb{Q}$ (`QQ` dans `Sage`), modèle type de corps infini, et de $\mathbb{K} = \mathbb{Z}/p\mathbb{Z}$ (`GF(p)` sous `Sage`) lorsque p est un nombre premier (tous les calculs étant alors conduits sous l'arithmétique modulo p). On peut également envisager le corps $\bar{\mathbb{Q}}$ (`QQbar` sous `Sage`) des nombres algébriques, c'est-à-dire des nombres complexes α solutions d'une équation polynomiale $a_0\alpha^d + \cdots + a_d = 0$ à coefficients $a_0, ..., a_d$ dans $\mathbb{Z}$ avec $a_0 \neq 0$; en effet, un tel nombre est identifié de manière exacte sous un logiciel de calcul formel tel `Sage` ; l'affichage de ce nombre correspond à une représentation « par intervalle » (suivie d'un point d'interrogation), mais il s'agit en fait d'une représentation qui pour le logiciel s'avère considérée comme exacte. Le corps $\mathbb{Q} \oplus i\mathbb{Q} = \mathbb{Q}[i]$ des nombres de Gauß pourra aussi s'avérer un modèle utile pour le calcul exact.

On pourra aussi envisager dans ce cours (c'est ce que l'on fera d'ailleurs le plus souvent) un corps $\mathbb{K}$ relevant du calcul scientifique (ou numérique) cette fois avec pertes : sous `Sage`, les exemples seront `RDF` et `CDF`(respectivement le corps des nombres réels et celui des nombres complexes, avec l'arithmétique approchée correspondant au

calcul *en double précision*[1]) ainsi que les modèles $\mathbb{K} =$ `RealField(Ndigits)` ou $\mathbb{K} =$ `ComplexField(Ndigits)`, `Ndigits` correspondant au nombre de « décimales » (en binaire : 0 ou 1) stockées après la virgule lorsque le nombre est exprimé en virgule flottante (pour plus de détails, on se reportera ici à la section 1.2 de [**CalcLog**]) ; notons que `RR` et `CC` correspondent à `Ndigits = 53`. L'intérêt de l'arithmétique sous `Sage` (si on la compare à celle opérée en simple ou double précision sous des logiciels de calcul scientifique tels `MATLAB` ou `Scilab` (dont la trame repose précisément sur le calcul matriciel, `MATLAB` signifiant par exemple « `Matrix Laboratory` ») est que la précision de la mantisse en virgule flottante peut être choisie arbitraire (avec la valeur arbitraire de `Ndigits`) alors qu'elle se trouve « figée » à 52 ou 23 (en fait 53 ou 24, mais un bit est stocké en mémoire) suivant que l'on travaille en double (`double`) ou simple (`single`) précision sous l'un ou l'autre de ces logiciels. Plutôt conçu pour des étudiants de la filière Mathématique-Informatique, ce cours sera essentiellement illustré sous `Sage` au détriment de `Scilab` ou `MATLAB`.

Exemple 1.1. À titre d'exemple, générons, lorsque `N` désigne un entier strictement positif, la matrice (symétrique) à entrées complexes $[\mathtt{W}_{\mathtt{N}}^{jk}]_{0\leq j,k\leq \mathtt{N}-1}$ (j indice de ligne, k indice de colonne), soit de manière exacte (`A`), soit de manière approchée (`Anum`).

```
sage:
corps = QQbar; W = exp(-2*i*pi/N)
A = Matrix(corps, [[W^(j*k) for k in range(N)]
           for j in range(N)])
sage:
corpsnum = ComplexField(Ndigits); W = exp(-2*i*pi/N)
Anum = Matrix(corpsnum, [[W^(j*k) for k in range(N)]
              for j in range(N)])
```

1.2. Matrices et applications linéaires, rang, génération aléatoire

Étant donnée une matrice à `M` lignes et `N` colonnes `A` à entrées dans un corps commutatif $\mathbb{K}$, on peut lui associer de manière naturelle une application $\mathbb{K}$-linéaire de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$ [2], à savoir l'unique application linéaire $L_{\mathtt{A}}$ telle que

$$L_{\mathtt{A}}(e_k) = (a_{0,k}, ..., a_{\mathtt{M}-1,k}), \quad k = 0, ..., \mathtt{M}-1$$

(les coordonnées du vecteur $L_{\mathtt{A}}(e_k)$ sont les entrées de la colonne d'indice k de `A`) si les vecteurs $e_0, ..., e_{\mathtt{N}-1}$ constituent la base canonique de $\mathbb{K}^{\mathtt{N}}$. On dit alors que *la matrice* `A` *représente l'application linéaire* $L_{\mathtt{A}}$ *lorsque les espaces respectivement source* $\mathbb{K}^{\mathtt{N}}$ *et but* $\mathbb{K}^{\mathtt{M}}$ *sont rapportés à leurs bases canoniques respectives.*

---

1. On consomme 64 bits pour coder les nombres réels en virgule flottante dans le système binaire : un bit pour le signe, 11 bits pour l'exposant, 52 bits pour la mantisse ($1 + 11 + 52 = 64 = 2^6$) ; en simple précision, la répartition est $1, 8, 23$ ($1 + 8 + 23 = 32 = 2^5$).

2. Il ne faut pas perdre de vue que le $\mathbb{K}$-espace vectoriel $\mathbb{K}^{\mathtt{N}}$ peut (ensemblistement) avoir bien des incarnations (toutes diverses) : l'ensemble dont les objets sont les vecteurs $(x_0, ..., x_{\mathtt{N}-1})$ à entrées dans le corps $\mathbb{K}$, mais aussi (ce qui est très fréquent en analyse numérique) celui dont les objets sont les fonctions définies sur l'ensemble fini $\{0, ..., \mathtt{N}-1\}$ et à valeurs dans $\mathbb{K}$, etc.

Les espaces source et but peuvent être rapportés à d'autres bases que leurs bases canoniques respectives. Si par exemple l'espace source $\mathbb{K}^{\mathtt{N}}$ est rapporté à la base $\{v_0, ..., v_{\mathtt{N}-1}\}$ et que l'espace but $\mathbb{K}^{\mathtt{M}}$ est rapporté à la base $\{w_0, ..., w_{\mathtt{M}-1}\}$, la matrice représentant l'application $L_{\mathtt{A}}$ lorsque les espaces source et but sont respectivement rapportés aux bases $\{v_0, ..., v_{\mathtt{N}-1}\}$ et $\{w_0, ..., w_{\mathtt{M}-1}\}$ s'obtient à partir de $\mathtt{A}$ par la transformation

$$\mathtt{A} \longmapsto \mathtt{Q}^{e_0,...,e_{\mathtt{M}-1}}_{w_0,...,w_{\mathtt{M}-1}} * \mathtt{A} * \mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}} \tag{1.1}$$

où :

- la *matrice de passage* $\mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}}$ (*de la base canonique de* $\mathbb{K}^{\mathtt{N}}$ *à la nouvelle base* $\{v_0, ..., v_{N-1}\}$ *de ce même espace vectoriel*) est la matrice carrée de taille $[\mathtt{N}, \mathtt{N}]$ dont les vecteurs colonne figurent (en colonne) les vecteurs v_j, $j = 0, ..., \mathtt{N} - 1$ de $\mathbb{K}^{\mathtt{N}}$ exprimés dans la base canonique $\{e_0, ..., e_{\mathtt{N}-1}\}$ de ce même espace source $\mathbb{K}^{\mathtt{N}}$;
- la *matrice de passage* $\mathtt{Q}^{e_0,...,e_{\mathtt{M}-1}}_{w_0...,w_{m-1}}$ (cette fois *de la nouvelle base* $\{w_0, ..., w_{M-1}\}$ *de* $\mathbb{K}^{\mathtt{M}}$ *à la base canonique de ce même espace vectoriel*) est la matrice carrée de taille $[\mathtt{M}, \mathtt{M}]$ dont les vecteurs colonne figurent (en colonne) les vecteurs e_j, $j = 0, ..., \mathtt{M}-1$ de la base canonique de $\mathbb{K}^{\mathtt{M}}$ exprimés dans la base $\{w_0, ..., w_{\mathtt{M}-1}\}$ de ce même espace but $\mathbb{K}^{\mathtt{M}}$;
- l'opération $*$ est le produit usuel de matrices, opération notée ainsi sous `sage` $(\mathtt{A}_0 * \mathtt{A}_1)$, licite seulement lorsque $\mathtt{A}_0$`.ncols()` $= \mathtt{A}_1$`.nrows()`.

On dit que les matrices $\mathtt{A}$ et

$$\mathtt{Q}^{e_0,...,e_{\mathtt{M}-1}}_{w_0,...,w_{\mathtt{M}-1}} * \mathtt{A} * \mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}}$$

sont des matrices *équivalentes*.

Remarque 1.1. Dans le cas particulier où $\mathtt{M} = \mathtt{N}$ et où les bases $\{v_0, ..., v_{\mathtt{N}-1}\}$ et $\{w_0, ..., w_{\mathtt{N}-1}\}$ coïncident, les deux matrices de passage $\mathtt{P}^{v_0,...,v_{n-1}}_{e_0,...,e_{\mathtt{N}-1}}$ et $\mathtt{Q}^{e_0,...,e_{\mathtt{N}-1}}_{v_0,...,v_{\mathtt{N}-1}}$ sont inverses l'une de l'autre (pour la multiplication $*$) et l'on peut donc reformuler la transformation (1.1) comme

$$\mathtt{A} \longmapsto (\mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}})^{-1} * \mathtt{A} * \mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}}. \tag{1.2}$$

On dit que les matrices $\mathtt{A}$ et

$$(\mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}})^{-1} * \mathtt{A} * \mathtt{P}^{v_0,...,v_{\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}}$$

sont deux matrices carrées `semblables`.

Étant donnée une matrice $\mathtt{A}$ à $\mathtt{M}$ lignes et $\mathtt{N}$ colonnes, l'action de cette matrice sur un vecteur v de $\mathbb{K}^{\mathtt{N}}$ (déclaré comme vecteur à partir de la liste de ses coordonnées suivant la syntaxe `vector(corps,liste)` sous `Sage`) se réalise aussi *via* l'opération notée `*`. Par exemple :

```
sage:
A = Matrix(QQ,[[-1/2,8,5,4],[3,1,-1/7,5],
           [4,1,0,-3/5],[0,1,0,4],[-1,1,2/5,0]])
V = vector(QQ,[1,-3/17,5/3,2]); W = A*V; W
reponse:
(1471/102, 4493/357, 223/85, 133/17, -26/51)
```

À toute application $\mathbb{K}$-linéaire de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$ est attachée une notion fondamentale, celle de *rang* : *le rang d'une application $\mathbb{K}$-linéaire L de $\mathbb{K}^{\mathtt{M}}$ dans $\mathbb{K}^{\mathtt{N}}$ est par définition la dimension du sous-espace* $\operatorname{Im} L$ *de l'espace but* $\mathbb{K}^{\mathtt{M}}$. C'est donc un entier compris entre 0 et $\min(\mathtt{M},\mathtt{N})$. En termes de matrices, on rebondit sur la notion suivante.

Définition 1.1 (rang d'une matrice). Le rang d'une matrice à `M` lignes et `N` colonnes `A` à entrées dans un corps commutatif $\mathbb{K}$ est par définition le rang de l'application linéaire $L_{\mathtt{A}}$ que cette matrice représente lorsque les $\mathbb{K}$-espaces vectoriels source et but $\mathbb{K}^{\mathtt{N}}$ et $\mathbb{K}^{\mathtt{M}}$ sont rapportés à leurs bases canoniques respectives.

Remarque 1.2 (conservation du rang par équivalence)**.** Toute matrice `B` déduite de `A` par la transformation (1.1), c'est-à-dire :

$$\mathtt{B} = \mathtt{Q}^{e_0,\dots,e_{\mathtt{M}-1}}_{w_0,\dots,w_{\mathtt{M}-1}} * \mathtt{A} * \mathtt{P}^{v_0,\dots,v_{\mathtt{N}-1}}_{e_0,\dots,e_{\mathtt{N}-1}}$$

(pour deux bases $\{v_0,...,v_{\mathtt{N}-1}\}$ et $\{w_0,...,w_{\mathtt{M}-1}\}$ respectivement de $\mathbb{K}^{\mathtt{N}}$ et de $\mathbb{K}^{\mathtt{M}}$) est telle que $\operatorname{rang}(\mathtt{B}) = \operatorname{rang}(\mathtt{A})$. On peut d'ailleurs montrer que deux matrices de même taille `[M,N]` sont équivalentes si et seulement si elles ont même rang.

Le rang d'une application $\mathbb{K}$-linéaire L de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$ est, on l'a mentionné, un entier inclus au sens large entre 0 et $\min(\mathtt{M},\mathtt{N})$. Il est important de préciser les deux cas « extrêmes » :

- si $\mathtt{N} \geq \mathtt{M}$, dire que $\operatorname{rang}(L) = \mathtt{M}$ équivaut à dire que L est une application $\mathbb{K}$-linéaire surjective de $\mathbb{K}^{\mathtt{N}}$ sur $\mathbb{K}^{\mathtt{M}}$;
- si $\mathtt{N} \leq \mathtt{M}$, dire que $\operatorname{rang} L = \mathtt{N}$ équivaut à dire que L est une application $\mathbb{K}$-linéaire injective de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$, ce qui signifie que son *noyau*, à savoir le sous-espace
$$\operatorname{Ker} L := \{v \in \mathbb{K}^{\mathtt{N}}\,;\, L(v) = 0\},$$
est le sous-espace $\{0\}$ de l'espace source $\mathbb{K}^{\mathtt{N}}$. On dispose d'ailleurs dans ce cas de l'importante *formule du rang* :

$$\mathtt{N} = \text{dimension de l'espace source} = \operatorname{rang}(L) + \dim_{\mathbb{K}^{\mathtt{N}}}(\operatorname{Ker} L). \tag{1.3}$$

Lorsque $\mathtt{M} = \mathtt{N}$, dire que $\operatorname{rang}(L) = \mathtt{M} = \mathtt{N}$ équivaut à dire que L réalise un isomorphisme de $\mathbb{K}^{\mathtt{N}} = \mathbb{K}^{\mathtt{M}}$, ce qui signifie qu'il existe une application $\mathbb{K}$-linéaire inverse $L^{-1} : \mathbb{K}^{\mathtt{N}} \to \mathbb{K}^{\mathtt{N}}$ telle que $L \circ L^{-1} = L^{-1} \circ L = \mathrm{Id}_{\mathbb{K}^{\mathtt{N}}}$.

Si `A` est une matrice carrée à `M` et `N` colonnes à entrées dans un corps commutatif $\mathbb{K}$, le rang de `A` est un entier compris au sens large entre 0 et $\min(\mathtt{M},\mathtt{N})$ on a donc (en revenant à l'application $\mathbb{K}$-linéaire L_A associée à `A`) les deux faits (extrêmes) suivants :

- si $\mathtt{N} \geq \mathtt{M}$, dire que $\operatorname{rang}(\mathtt{A}) = \mathtt{M}$ équivaut à dire que
$$v \in \mathbb{K}^{\mathtt{N}} \to w = A * v \in \mathbb{K}^{\mathtt{M}}$$
est une application surjective, ou encore qu'étant donné un vecteur $y = (y_0,...,y_{\mathtt{M}-1})$ quelconque de l'espace but $\mathbb{K}^{\mathtt{M}}$, il existe au moins un vecteur x de l'espace source $\mathbb{K}^{\mathtt{N}}$ tel que $y = \mathtt{A} * x$, ce qui s'exprime aussi en disant que le système de `M` équations linéaires en `N` inconnues avec second membre
$$\sum_{k=0}^{\mathtt{N}} a_{j,k}\, x_k = y_j \quad (j = 0,...,\mathtt{M}-1)$$

admet au moins une solution $x = (x_0, ..., x_{\mathtt{N}-1})$ dans $\mathbb{K}^{\mathtt{N}}$ (ou encore est *compatible*) ;

— si $\mathtt{N} \leq \mathtt{M}$, dire que $\mathrm{rang}\,(\mathtt{A}) = \mathtt{N}$ équivaut à dire que

$$v \in \mathbb{K}^{\mathtt{N}} \to w = A * v \in \mathbb{K}^{\mathtt{M}}$$

est une application linéaire injective, ou encore que le système homogène de $\mathtt{M}$ équations à $\mathtt{N}$ inconnues

$$\sum_{k=0}^{\mathtt{N}} a_{j,k}\, x_k = 0 \quad (j = 0, ..., \mathtt{M}-1)$$

n'admet comme seule solution que $x = (0, ..., 0) = 0_{\mathbb{K}^{\mathtt{N}}}$.

Lorsque $\mathtt{M} = \mathtt{N}$, dire que $\mathrm{rang}(\mathtt{A}) = \mathtt{N} = \mathtt{M}$ équivaut à dire que $\mathtt{A}$ est une matrice carrée inversible, ce qui signifie qu'il existe une matrice carrée $\mathtt{A}^{-1}$ telle que $\mathtt{A} * \mathtt{A}^{-1} = \mathtt{A}^{-1} * \mathtt{A} = \mathtt{Id}_{\mathtt{N}}$, où $\mathtt{I}_{\mathtt{N}} = \mathtt{identity\_matrix(N)}$ désigne la matrice identité de taille $(\mathtt{N}, \mathtt{N})$ (ainsi identifiée sous `Sage` [1]).

Lorsque le corps $\mathbb{K}$ est un corps dans lequel `Sage` opère comme un logiciel de calcul formel (par exemple $\mathbb{K} = \mathbb{Q} = \mathtt{QQ}$, $\bar{\mathbb{Q}} = \mathtt{QQbar}$, $\mathbb{Z}/p\mathbb{Z} = \mathtt{GF(p)}$ avec p premier), la commande `A.rank()` retourne le rang d'une matrice `A` à entrées dans $\mathbb{K}$. Cette commande est inefficace lorsque $\mathbb{K}$ constitue un environnement dans lequel les calculs sont des calculs approchés (`RDF` ou `RealField(Ndigits)` pour le corps $\mathbb{R}$ version « numérique », `CDF` ou `ComplexField(Ndigits)` pour le corps $\mathbb{C}$ version « numérique »).

Souvent dans ce cours, nous aurons à générer des matrices de manière aléatoire. Ce pourront être

— des matrices binaires (c'est-à-dire dont les entrées sont des constituées de 0 ou de 1) ;

— des matrices à coefficients dans $\mathbb{Z}$ (les entrées étant spécifiées appartenir à $\{a, a+1, ..., b-1, b\}$, où a et b sont des entiers naturels tels que $a < b$) ;

— des matrices à coefficients réels approchés ou complexes approchés (avec une précision de `Ndigits`) appartenant au segment réel $[a, b]$ ($a < b$ avec $a, b \in \mathbb{R}$) ou au pavé du plan complexe

$$\{\mathrm{Re}\, z \in [a, b],\ \mathrm{Im}\, z \in [c, d]\},$$

où a, b, c, d sont des nombres réels tels que $a < b$ et $c < d$.

Le traitement de ces matrices relèvera dans les deux premiers cas du calcul symbolique, dans le troisième du calcul scientifique. Voici (dans chacun des trois cas) le mode de génération sous `Sage` (pour des matrices à $\mathtt{M}$ lignes et $\mathtt{N}$ colonnes) :

```
sage:
A_bin = Matrix(ZZ,[[ZZ.random_element(x=0,y=2)
                for k in range(N)] for j in range(M)])
```

1. Il arrive (sous certaines versions non actualisées du logiciel) que ce soit plutôt la commande `matrix.identity(N)` qui rétourne cette matrice identité, élément unité (à gauche et à droite) dans l'anneau des matrices carrées à coefficients dans $\mathbb{K}$ de taille `[N,N]`.

```
sage:
A_entiers = Matrix(ZZ,[[ZZ.random_element(x=a,y=b+1)
             for k in range(N)] for j in range(M)])

sage:
corpsR = RealField(Ndigits)
A_reels = Matrix(corpsR,[[corpsR.random_element(a,b)
                 for k in range(N)] for j in range(M)])

sage:
corpsC = ComplexField(Ndigits)
corpsR = RealField(Ndigits)
A_complexes = Matrix(corpsC,[[corpsR.random_element(a,b)
                 + i*corpsR.random_element(c,d)
                 for k in range(N)] for j in range(M)])
```

Dans tous les cas ci-dessus, la génération aléatoire obéit à la loi uniforme. Pour les deux derniers exemples (numériques), il peut s'avérer intéressant (en particulier pour modéliser les erreurs numériques, on prendra dans ce cas `moyenne=0`) de remplacer la génération aléatoire suivant une distribution uniforme sur un segment $[a, b]$ de $\mathbb{R}$ ou un pavé $[a, b] \times [c, d]$ de $\mathbb{C}$ par la génération suivant une loi de probabilité gaussienne de moyenne `moyenne` et d'écart-type `std` (*standard deviation*) dans le cas réel, de vecteurs des moyennes `(moyenne1,moyenne2)` et de vecteur des écarts-types `(std1,std2)` dans le cas complexe. Cela donne dans les deux cas :

```
sage:
corpsR = RealField(Ndigits)
A_greels = Matrix(corpsR,M,N,lambda j,k:normalvariate(moyenne,std))

sage:
corpsC = ComplexField(Ndigits)
corpsR = RealField(Ndigits)
A_gcomplexes = Matrix(corpsR,M,N,
               lambda j,k:normalvariate(moyenne1,std1))
+i*Matrix(corpsR,M,N,
          lambda j,k:normalvariate(moyenne2,std2))
```

On pourra tester que, dans le contexte numérique (matrices aléatoires à entrées réelles ou complexes), la probabilité que le rang de la matrice ainsi générée soit différent de $\min(\mathtt{M}, \mathtt{N})$ est nulle.

1.3. $\mathbb{K}$-espace vectoriel des matrices de taille fixée ; ≪ dot ≫ produit

Les matrices à `M` lignes et `N` colonnes à entrées dans un corps $\mathbb{K}$ forment un $\mathbb{K}$-espace vectoriel $\mathscr{M}(\mathbb{K}; \mathtt{M}, \mathtt{N})$, tout comme celui que forment les applications $\mathbb{K}$-linéaires de $\mathbb{K}^{\mathtt{M}}$ dans $\mathbb{K}^{\mathtt{N}}$ en correspondance avec $\mathscr{M}(\mathbb{K}; \mathtt{M}, \mathtt{N})$ via la correspondance $\mathtt{A} \longleftrightarrow L_{\mathtt{A}}$ mentionnée à la section précédente.

Sous `Sage`, ce $\mathbb{K}$-espace vectoriel se déclare ainsi ($\mathbb{K} =$ `corps`) :

```
sage:
Mat = MatrixSpace(corps,M,N)
```

Une autre opération (cette fois interne) dans $\mathscr{M}(\mathbb{K};\mathtt{M},\mathtt{N})$ joue un rôle important principalement en calcul scientifique, dans les logiciels à base de langage « interprété » tels `MATLAB` et `Scilab` : il s'agit du « dot produit » `A.*B`, dit encore *produit de Hadamard*[1], consistant à multiplier deux matrices de même taille (à coefficients dans le corps $\mathbb{K}$) en en multipliant les entrées terme à terme. Sous `Sage`, cette opération, étrangère au cadre de l'algèbre linéaire mais néanmoins très importante du point de vue pratique, se réalise ainsi :

```
sage :
def dotPRODUCT(A,B):
    return A.parent()([a*b for a,b
                       in zip(A.list(),B.list())])
```

1.4. Normes matricielles ou non sur $\mathscr{M}(\mathbb{K};\mathtt{M},\mathtt{N})$, exemples

Dans cette section, $\mathbb{K}$ désigne soit le corps des nombres rationnels $\mathbb{Q}$ (`QQ` sous `Sage`) ou bien le corps $\overline{\mathbb{Q}}$ (`QQbar` sous `Sage`) dans le contexte du calcul formel, soit une version numérique du corps des nombres réels $\mathbb{R}$ ou du corps des nombres complexes $\mathbb{C}$ dans le contexte du calcul scientifique, c'est-à-dire l'un des environnements numériques `RDF`, `CDF`, `RIF`, `CIF`, `RealField(Ndigits)`, `ComplexField(Ndigits)`, etc. si l'on envisage de travailler sous `Sage`. Notre point de vue dans cette section sera plutôt celui du calcul scientifique.

La norme (notée $|\ |$) sur $\mathbb{K}$ (ici $\mathbb{K}=\mathbb{R}$ ou $\mathbb{C}$, voire $\mathbb{Q}$ ou $\overline{\mathbb{Q}}$ considérés comme des sous-corps des précédents), désignera ici la norme usuelle, dite *archimédienne*; cette norme vérifie en effet l'inégalité triangulaire $|x+y|\leq|x|+|y|$ pour tout x,y dans le corps $\mathbb{K}$[2]. Une fois cette norme $|\ |$ fixée sur $\mathbb{K}$, le choix d'une norme sur $\mathbb{K}^{\mathtt{N}}$ ($\mathbb{K}=\mathbb{R}$ ou $\mathbb{C}$) a pu jusque là paraître indifférent. Lorsque $\mathbb{K}=\mathbb{R}$ ou $\mathbb{C}$ (ce sont tous deux ici des *corps métriques complets*, c'est-à-dire des corps dans lesquels toute suite de Cauchy converge, ou encore, ce qui est équivalent, toute série absolument convergente est automatiquement convergente) toutes les normes (archimédiennes) sur un $\mathbb{K}$-espace de dimension finie (ici $\mathbb{K}^{\mathtt{N}}$) sont en effet équivalentes, ce qui implique qu'étant données deux normes $\|\ \|$ et $\|\ \|'$ sur $\mathbb{K}^{\mathtt{N}}$, il existe toujours une constante $K>0$ telle que l'on ait l'encadrement $\|\ \|/K\leq\|\ \|'\leq K\|\ \|$. Changer de norme dans $\mathbb{K}^{\mathtt{N}}$ n'est cependant pas tout-à-fait aussi anodin que l'on pourrait le penser. Choisir une norme sur $\mathbb{K}^{\mathtt{N}}$, c'est en effet choisir une manière de « mesurer » la « taille » des vecteurs et, par voie

1. On retrouvera plusieurs fois dans ce cours le nom de Jacques Hadamard. Mathématicien français, 1865-1963, il s'illustra par ses travaux en analyse, mais aussi par ses contributions à la cryptographie. C'est à la faculté des sciences de Bordeaux qu'il débuta (entre 1893 et 1896) sa carrière d'enseignant et de chercheur.

2. Notons qu'une norme sur un corps commutatif $\mathbb{K}$ est dite *ultramétrique* ou *non-archimédienne* si, en place de l'inégalité triangulaire, elle vérifie l'inégalité *ultramétrique* $|a+b|\leq\max(|a|,|b|)$. Tel est le cas de la norme p-adique (p désignant un nombre premier) sur $\mathbb{Q}$: $|a/b|_p:=p^{-\nu_p(|a|)+\nu_p(|b|)}$, où $\nu_p(m)$ désigne l'exposant de p dans la décomposition en facteurs premiers d'un entier strictement positif m (avec de plus $\nu_p(0)=+\infty$). Le complété de $\mathbb{Q}$ pour cette norme p-adique est le corps $\mathbb{Q}_p$ des nombres p-adiques. Notons que l'on peut travailler dans $\mathbb{Q}_p$ sous `Sage`; le corps $\mathbb{Q}_p$ se déclare en effet comme `Qp(p)` sous ce logiciel.

de conséquence, celle des opérateurs de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$, l'action de ces opérateurs étant représentée par leur matrice rapportée au choix de la base canonique à la source ($\mathbb{K}^{\mathtt{N}}$) et au but ($\mathbb{K}^{\mathtt{M}}$).

Les normes les plus importantes que l'on puisse choisir sur $\mathbb{K}^{\mathtt{N}}$ sont [1] :

— la norme euclidienne $\|\ \|_2$

$$\|(x_0,...,x_{\mathtt{N}-1})\|_2 := (x_0^2+\cdots+x_{\mathtt{N}-1}^2)^{1/2}\,,$$

importante car liée au produit scalaire [2] dans $\mathbb{K}^{\mathtt{N}}$, défini, lui, par

$$\langle x,y\rangle = \sum_{j=0}^{\mathtt{N}-1} x_j\bar{y}_j = \mathtt{Y}^*\ast\mathtt{X}, = \texttt{Y.conjugate()*X},$$

outil permettant de faire dans $\mathbb{K}^N$ précisément de la géométrie « à la Pythagore » et d'y développer ce que nous appellerons l'*algorithmique hilbertienne*, nous y reviendrons ultérieurement ; c'est souvent la norme par défaut `norm` de la plupart des logiciels de calcul scientifique. Sous `Sage`, on la déclare par :

```
sage:
V = vector(RealField(53),[1,4,5,7]); V.norm(2)
ans:
9.53939201416946
```

(on pourrait alternativement utiliser `sqrt(V.conjugate()*V)` pour la retourner).

— la norme $\|\ \|_\infty$ définie par

$$\|(x_0,...,x_{\mathtt{N}-1})\|_\infty := \max_{j=0,...,\mathtt{N}-1}|x_j|\,; \tag{1.4}$$

cette norme (qui n'est, au contraire de la norme euclidienne, que continue sur $\mathbb{K}^{\mathtt{N}}\setminus\{0\}$ alors que la norme euclidienne présente l'avantage, elle, d'être C^∞ sur cet ouvert $\mathbb{K}^{\mathtt{N}}\setminus\{0\}$) se déclare sous `Sage` ainsi :

```
sage:
V = vector(RealField(53),[1,4,5,2*pi]); V.norm(infinity)
ans:
6.28318530717959
```

— la norme $\|\ \|_1$ définie par

$$\|(x_0,...,x_{\mathtt{N}-1})\|_1 := \sum_{j=0}^{\mathtt{N}-1}|x_j|\,;$$

cette norme se déclare sous `Sage` suivant la syntaxe :

```
sage:
V = vector(RealField(53),[1,4,5,2*pi]); V.norm(1)
ans:
16.2831853071796
```

1. Nous convenons d'utiliser ici les conventions d'indexation de `Python`.
2. On parle aussi de *corrélation discrète* dans le cadre de la théorie de l'information.

— plus généralement, si p désigne un nombre réel supérieur [1] ou égal 1, la norme $\| \ \|_p$ définie par [2]

$$\|(x_0, ..., x_{\mathtt{N}-1})\|_p := \Big(\sum_{j=0}^{\mathtt{N}-1} |x_j|^p \Big)^{1/p} .$$

Sous `Sage`, on doit par exemple déclarer cette norme ainsi (par exemple lorsque `Ndigits` $= 53$ et $p = \sqrt{2}$) :

```
sage:
V = vector(RealField(53),[1,4,5,2*pi])
V.norm(RealField(53)(sqrt(2)))
ans:
11.4142897138019
```

Le choix d'une norme $\| \ \|$ sur $\mathbb{K}^{\mathtt{N}}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) induit le choix d'une norme sur le $\mathbb{K}$-espace vectoriel des applications linéaires $L : \mathbb{K}^{\mathtt{N}} \longmapsto \mathbb{K}^{\mathtt{M}}$ par

$$\|L\| := \sup_{X \in \mathbb{K}^{\mathtt{N}} \setminus \{0\}} \frac{\|L(X)\|}{\|X\|} . \tag{1.5}$$

Une telle norme (sur le $\mathbb{K}$-espace des applications linéaires de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$) est dite *norme d'opérateur* attachée à la norme $\| \ \|$ dont elle est déduite. Comme se donner un opérateur L revient à ce donner la matrice `A` de cet opérateur lorsque les $\mathbb{K}$-espaces vectoriels source ($\mathbb{K}^{\mathtt{N}}$) et but ($\mathbb{K}^{\mathtt{M}}$) sont rapportés à leurs bases canoniques respectives ($L = L_{\mathtt{A}}$), on appelle une telle norme d'opérateur *norme matricielle* à partir du moment où l'on envisage cette norme non plus sur le $\mathbb{K}$-espace des opérateurs $\mathbb{K}$-linéaires de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$, mais (ce qui revient de fait au même) sur les matrices (ou tableaux bidimensionnels) `A` à `M` lignes et `N` colonnes.

Dans le cas où $p = 1$ et $p = \infty$, la norme matricielle déduite de la norme de Minkowski $\| \ \|_p$ est aisée à représenter :

— dans le cas $p = 1$, la norme $\|\mathtt{A}\|_1$ d'une matrice $\mathtt{A} = [a_{j,k}]_{0 \le j \le \mathtt{M}-1, 0 \le k \le \mathtt{N}-1}$ à `M` lignes et `N` colonnes est

$$\|\mathtt{A}\|_1 = \max_k \sum_{j=0}^{\mathtt{M-1}} |a_{j,k}|. \tag{1.6}$$

puisque

$$\sum_{j=0}^{\mathtt{M-1}} \Big| \sum_{k=0}^{\mathtt{N}-1} a_{j,k}\, x_k \Big| \le \Big(\max_k \Big(\sum_{j=0}^{\mathtt{M}-1} |a_{j,k}| \Big) \Big) \times \|x\|_1 \tag{1.7}$$

1. Pour $p \in]0, 1[$, l'inégalité triangulaire est en défaut ; on dispose même d'ailleurs d'une version ≪ renversée ≫de l'inégalité triangulaire dans ce cas ; nous n'avons donc plus affaire alors à une norme. Cependant, la norme $\| \ \|_0$ (qui en fait n'est pas une norme) peut être appelée à jouer un rôle important, surtout lorsque les questions de *parcimonie* sont en jeu ; par définition $\|x\|_0$ dénote le nombre d'entrées non nulles du vecteur $x \in \mathbb{K}^{\mathtt{N}}$.

2. Qu'il s'agisse d'une norme est ici moins immédiat ; l'inégalité triangulaire est une célèbre inégalité due au mathématicien russe, géomètre tant des espaces que des nombres, Hermann Minkowski, 1864-1909.

pour tout $x \in \mathbb{K}^{\mathtt{N}}$ et que la constante

$$\max_k \Big(\sum_{j=0}^{\mathtt{M}-1} |a_{j,k}| \Big)$$

est la plus petite à satisfaire l'inégalité (1.7) ;

— dans le cas $p = \infty$, la norme $\|\mathtt{A}\|_\infty$ d'une matrice à $A = [a_{j,k}]_{0\le j\le \mathtt{M}, 0\le k\le \mathtt{N}-1}$ à `M` lignes et `N` colonnes est

$$\|\mathtt{A}\|_\infty = \max_j \sum_{k=0}^{\mathtt{N}-1} |a_{j,k}| \tag{1.8}$$

puisque

$$\max_j \big| \sum_{k=0}^{\mathtt{N}-1} a_{j,k}\, x_k \big| \le \Big(\max_j \Big(\sum_{k=0}^{\mathtt{N}-1} |a_{j,k}| \Big) \Big) \times \|x\|_\infty \tag{1.9}$$

pour tout $x \in \mathbb{K}^{\mathtt{N}}$ et que la constante

$$\max_j \sum_{k=0}^{\mathtt{N}-1} |a_{j,k}|$$

est, une fois encore, la plus petite à satisfaire l'inégalité (1.9).

Notons que, sous `Sage`, l'affichage de l'évaluation « numérique » de la norme $\| \ \|_p$ d'une matrice `A` n'est envisageable que si $p = 1, 2, \infty$ et se fait systématiquement en double précision sans prendre en ligne de compte la précision `Ndigits` avec laquelle aurait pu être déclarée éventuellement la matrice `A` .

On reviendra plus loin dans le cours sur le calcul de la norme matricielle $\| \ \|_2$ d'une matrice de taille `[M,N]` à coefficients réels ou complexes car la norme matricielle $\| \ \|_2$ est appelée à jouer un rôle très important en relation avec le concept d'orthogonalité ; contentons nous de dire pour l'instant que $\|\mathtt{A}\|_2$ est définie comme la racine carrée de la plus grande des valeurs propres (toutes réelles, positives ou nulles) de la matrice symétrique réelle $\mathtt{A}^* * \mathtt{A}$ ($\mathtt{A}^*$, notée aussi $\mathtt{A}'$ dans les logiciels de calcul scientifique, étant la trans-conjuguée –ou adjointe [1]– de `A`, c'est-à-dire la conjuguée de la transposée de cette matrice `A`, ces deux opérations commutant entre elles).

Étant données deux matrices carrées `A` et `B` de même taille `[N,N]` et un choix de norme dans $\mathbb{K}^{\mathtt{N}}$, on a toujours, pour la norme matricielle induite sur le $\mathbb{K}$-espace vectoriel des matrices de taille `[N,N]`, l'inégalité de sous-multiplicativité :

$$\|\mathtt{A} * \mathtt{B}\| \le \|\mathtt{A}\| \times \|\mathtt{B}\| . \tag{1.10}$$

Si `A` est une matrice à `M` lignes et `N` colonnes, `B` une matrice à `N` lignes et `NN` colonnes et que le même choix de norme est effectué sur $\mathbb{K}^{\mathtt{M}}$, $\mathbb{K}^{\mathtt{N}}$ et $\mathbb{K}^{\mathtt{NN}}$, l'inégalité (1.10) subsiste d'ailleurs (si $\| \ \|$ désigne chaque fois la norme matricielle attachée à ce choix de normes sur les espaces vectoriels $\mathbb{K}^{\mathtt{M}}$, $\mathbb{K}^{\mathtt{N}}$ et $\mathbb{K}^{\mathtt{NN}}$).

---

1. On dispose en effet dans $\mathbb{K}^{\mathtt{N}}$ de la formule dite « d'adjonction » : si v et w sont deux vecteurs de $\mathbb{K}^{\mathtt{N}}$, on a

$$(\mathtt{A} * \mathtt{v}) * \mathtt{w} = \mathtt{v} * (\mathtt{A}^* * \mathtt{w}) = \texttt{v*(((A*transpose()).conjugate())*w)}.$$

Ceci éclaire cette terminologie de « matrice adjointe ».

En revanche, il existe des normes sur le $\mathbb{K}$-espace des matrices de taille `[M,N]` à entrées dans $\mathbb{K}$ qui ne sont pas sous-multiplicatives lorsque $\mathtt{M} = \mathtt{N}$; tel est par exemple le cas de la norme (pourtant importante du point de vue pratique car aisée à définir) définie par

$$\|\mathtt{A}\|_{\sup} = \max_{j,k} |a_{j,k}|.$$

Pareilles normes ne sauraient donc être des normes matricielles. Une autre telle norme importante (elle aussi non matricielle) sur l'espace des matrices à coefficients dans $\mathbb{K}$ de taille `[M,N]` est la *norme de Frobenius* $\|\mathtt{A}\|_{\mathrm{Frob}}$ définie par

$$\|\mathtt{A}\|_{\mathrm{Frob}} = \sqrt{\mathrm{Trace}[\mathtt{A}^* * \mathtt{A}]} \tag{1.11}$$

où $\mathtt{A}^*$ désigne toujours la *trans-conjuguée* de la matrice `A`. La *trace* d'une matrice carrée est définie, on le verra dans la section suivante, comme la somme des éléments diagonaux de cette matrice et ne dépend que de la classe de similarité de la matrice en question (deux matrices carrées semblables ont même trace).

1.5. Déterminant, polynôme caractéristique et valeurs propres

Introduite par le mathématicien suisse Gabriel Cramer dès 1750, puis approfondie par des mathématiciens britanniques tels Arthur Cayley, 1821-1895, James Sylvester, 1814-1897, etc., enrichie ensuite extensivement par d'autres tels Francis Sowerby Macaulay, 1862-1937 (un logiciel de calcul formel porte aujourd'hui son nom), la notion de *déterminant* joue un rôle majeur (du point de vue théorique) en algèbre linéaire et en théorie de l'élimination [1] . Il convient toutefois de mentionner que le calcul du déterminant d'une matrice carrée `A` à entrées dans $\mathbb{K}$ (opéré sous `Sage` par la règle `A.determinant()`), calcul réalisé grâce aux développements successifs suivant la règle de Sarrus, s'avère extrêmement couteux : le déterminant d'une matrice $[a_{j,k}]_{0\leq j,k\leq \mathtt{N}-1}$ se présente en effet sous la forme d'une combinaison de `N`! termes

$$\pm a_{j_1,k_1} \cdots a_{j_n,k_n},$$

où les a_{j_ℓ,k_ℓ} sont des entrées de $\mathbb{K}$ (une et une seule par colonne) ; or `N`! devient très vide de taille énorme ; par exemple :

```
sage: factorial(30)
reponse: 265252859812191058636308480000000
```

Du point de vue algorithmique, le calcul de déterminant se doit en général d'être évité (donc contourné), ce que nous n'aurons de cesse de faire tout au long de ce cours.

L'un des objets fondamentaux associés à une matrice carrée (de taille `[N,N]`) à entrées dans un corps $\mathbb{K}$ est son *polynôme caractéristique*.

Définition 1.2 (polynôme caractéristique d'une matrice `A` ou de l'endomorphisme de $\mathbb{K}^{\mathtt{N}}$ correspondant $L_{\mathtt{A}}$). Si `A` est une matrice carrée de taille `[N,N]`, on appelle *polynôme caractéristique* de `A` l'élément de $\mathbb{K}[\mathtt{X}]$ défini par

$$\Phi_{\mathtt{A}}(\mathtt{X}) = \det(\mathtt{X} * \mathtt{identity\_matrix}(\mathtt{N}) - \mathtt{A}). \tag{1.12}$$

1. Il faut aussi mentionner que le déterminant d'un système de vecteurs $(v_0, ..., v_{\mathtt{N}-1})$ formant une base de $\mathbb{R}^n$ a pour valeur absolue le volume euclidien du parallélotope de l'espace affine $\mathbb{R}^{\mathtt{N}}$ construit sur ces vecteurs, d'où le rôle important des déterminants en analyse (théorie de l'intégration) et en géométrie.

Sous `Sage`, cet objet se déclare ainsi ($\mathbb{K}$ = `corps`) :

```
sage:
R.<X> = corps[]; Phi_A = A.charpoly(var='X')
```

Remarque 1.3. Si `A` et `B` sont deux matrices semblables (représentant le même endomorphisme de $\mathbb{K}^{\mathtt{N}}$ lorsque $\mathbb{K}^{\mathtt{N}}$ est rapporté à deux bases différentes), les polynômes caractéristiques $\Phi_{\mathtt{A}}$ et $\Phi_{\mathtt{B}}$ coïncident, ce qui fait que l'on peut aussi dire que $\Phi_{\mathtt{A}}$ est le polynôme caractéristique de l'endomorphisme $L_{\mathtt{A}}$ de $\mathbb{K}^{\mathtt{N}}$ que représente `A` lorsque $\mathbb{K}^{\mathtt{N}}$ est rapporté (à la source et au but) à sa base canonique.

Les coefficients du polynôme caractéristique d'une matrice carrée `A` à coefficients dans un corps $\mathbb{K}$ ne dépendent (d'après la remarque 1.3 ci dessus) que de l'endomorphisme $L_{\mathtt{A}}$ de $\mathbb{K}^{\mathtt{N}}$ que cette matrice représente lorsque $\mathbb{K}^{\mathtt{N}}$ est rapporté à sa base canonique à la source et au but ; certains coefficients (comme la trace [1] `A.trace()` ou le déterminant `A.determinant()`) jouent un rôle important dans la connaissance de l'endomorphisme $L_{\mathtt{A}}$: on a

$$\Phi_{\mathtt{A}}(\mathtt{X}) = \mathtt{X}^{\mathtt{N}} - \mathtt{A.trace()}\,\mathtt{X}^{\mathtt{N}-1} + \cdots + (-1)^{\mathtt{N}}\,\mathtt{A.determinant()}.$$

Dans le cas particulier où $\mathbb{K} = \mathbb{Q}$, $\mathbb{K} = \mathbb{R}$ ou $\mathbb{K} = \mathbb{C}$, on sait d'après le théorème fondamental de l'algèbre que le polynôme caractéristique (unitaire, c'est-à-dire de coefficient dominant égal à 1) $\Phi_{\mathtt{A}}$ d'une matrice carrée à coefficients dans $\mathbb{K}$ (ou, ce qui revient d'après ce qui précède au même) d'une application $\mathbb{K}$-linéaire de $\mathbb{K}^{\mathtt{N}}$ dans lui-même (on dit aussi un endomorphisme de $\mathbb{K}^{\mathtt{N}}$) se présente sous forme scindée :

$$\Phi_{\mathtt{A}}(\mathtt{X}) = \prod_{j=0}^{\mathtt{N}-1}(X - \lambda_j)$$

où $\lambda_0, ..., \lambda_{\mathtt{N}-1}$ sont `N` nombres complexes. Lorsque $\mathbb{K} = \mathbb{Q}$ ou $\mathbb{K} = \mathbb{R}$, ceux parmi ces nombres qui ne sont pas réels figurent dans la liste par paires $(\lambda, \bar{\lambda})$, où $\bar{\lambda} = \alpha - i\beta$ ($\alpha \in \mathbb{R}, \beta \in \mathbb{R}^*$) est le conjugué de $\lambda = \alpha + i\beta$.

Définition 1.3 (valeurs propres complexes d'une matrice carrée `A` ou d'un endomorphisme $L_{\mathtt{A}}$ lorsque le corps de référence est $\mathbb{Q}, \mathbb{R}$ ou $\mathbb{C}$, spectre et rayon spectral de `A` ou $L_{\mathtt{A}}$). On appelle *valeur propre* (complexe) d'une matrice carrée `A` à entrées dans $\mathbb{Q}, \mathbb{R}$ ou $\mathbb{C}$ toute racine (dans $\mathbb{C}$) de son polynôme caractéristique $\Phi_{\mathtt{A}}$. Le sous-ensemble (fini, de cardinal au plus `N` si `A` est de taille `[N,N]`) de $\mathbb{C}$ constitué des valeurs propres de `A` est appelé *spectre complexe* de `A` (ou spectre complexe de l'endomorphisme $L_{\mathtt{A}}$), tandis que le nombre

$$\rho(\mathtt{A}) \;:= \max_{0\le j\le \mathtt{N}-1} |\lambda_j| \ge 0$$

(rayon du plus petit disque de $\mathbb{C}$ de centre l'origine qui contiennent toutes les valeurs propres de `A`) est appelé *rayon spectral* de `A` (ou de l'endomorphisme $L_{\mathtt{A}}$).

Cette définition s'accompagne de la suivante :

1. La *trace* d'une matrice carrée figure la somme de ses termes diagonaux ; deux matrices carrées semblables ont même trace, puisqu'elles ont même polynôme caractéristique, donc aussi même déterminant, etc.

DÉFINITION 1.4 (sous-espaces propres complexes, vecteurs propres complexes, d'une matrice `A` à coefficients dans $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ ou de l'endomorphisme $L_{\mathtt{A}}$ correspondant). Dire que $\lambda \in \mathbb{C}$ est une valeur propre d'une matrice carrée `A` à entrées dans $\mathbb{K} = \mathbb{Q}, \mathbb{R}$ ou $\mathbb{C}$ (ou de l'endomorphisme $L_{\mathtt{A}}$ que `A` représente lorsque $\mathbb{K}^{\mathtt{N}}$ est rapporté à sa base canonique) équivaut à dire que le sous-espace $\mathrm{Ker}(\lambda \mathrm{Id}_{\mathtt{N}} - L_{\mathtt{A}})$ de $\mathbb{C}^{\mathtt{N}}$ est de dimension strictement positive. On appelle ce sous-espace $\mathrm{Ker}(\lambda \mathrm{Id}_{\mathtt{N}} - L_{\mathtt{A}})$ de $\mathbb{C}^{\mathtt{N}}$ le *sous-espace propre* complexe attaché à la valeur propre λ ; les vecteurs non nuls de ce sous-espace propre $\mathrm{Ker}(\lambda \mathrm{Id}_{\mathtt{N}} - L_{\mathtt{A}}) \subset \mathbb{C}^{\mathtt{N}}$ sont appelés *vecteurs propres* de $L_{\mathtt{A}}$ (dans $\mathbb{C}^{\mathtt{N}}$) (attachés à cette valeur propre complexe λ).

Remarque 1.4 (valeurs propres de `A` dans $\mathbb{K}$ et sous-espaces propres attachés dans $\mathbb{K}^{\mathtt{N}}$)**.** Si λ est une valeur propre de `A` appartenant au corps $\mathbb{K}$, le sous-espace $\mathrm{Ker}_{\mathbb{K}}(\lambda \mathrm{Id}_{\mathtt{N}} - L_{\mathtt{A}})$ (considéré comme un sous-espace vectoriel cette fois de $\mathbb{K}^{\mathtt{N}}$ et non plus de $\mathbb{C}^{\mathtt{N}}$) est de dimension strictement positive. On l'appelle sous-espace propre (attaché à la valeur propre λ de $L_{\mathtt{A}}$ dans $\mathbb{K}$) de $L_{\mathtt{A}}$ dans $\mathbb{K}^{\mathtt{N}}$ et ses vecteurs non nuls sont dits *vecteurs propres* de $L_{\mathtt{A}}$ (ou de `A`) dans $\mathbb{K}^{\mathtt{N}}$ (attachés à cette valeur propre $\lambda \in \mathbb{K}$).

Lorsque $\mathbb{K} = \mathbb{Q}$ ou $\mathbb{K} = \bar{\mathbb{Q}}$ (`QQ` ou `QQbar` sous `Sage`) et que `A` est une matrice carrée à entrées rationnelles ou algébriques (suivant le cas), les valeurs propres complexes de `A` , ainsi que les multiplicités dont sont affectées ces valeurs propres comme zéros du polynôme caractéristique $\Phi_{\mathtt{A}}$, sont retournées exactement sous `Sage` avec cette ligne de commandes :

```
sage:
Phi = A.charpoly()
Phi.roots(QQbar)
```

Les valeurs propres sont ici affichées avec une représentation des nombres complexes « par intervalle » (`ComplexIntervalField(Ndigits)`) mais il s'agit pourtant ici d'une représentation exacte. Voici un exemple :

```
sage:
A = Matrix(QQ,[[3,2,1],[2,-1,-2],[1,-2,4]])
Phi=A.charpoly(); Phi.roots(QQbar)
reponse:
[(-2.584428340330492?, 1), (3.805118086952745?, 1),
(4.779310253377747?,1)]
```

Mais lorsque le corps $\mathbb{K}$ est modélisé par un environnement numérique (`RDF` ou `CDF`, `RealField(Ndigits)` ou `ComplexField(Ndigits)`), le calcul même du polynôme caractéristique peut se trouver, par exemple dès que la norme $\|\mathtt{A}\|_{\sup}$ est grande, entaché d'erreurs d'arrondi rendant le retour des valeurs propres sous `Sage` lui aussi entaché d'erreurs grossières. On observe ceci par exemple si l'on travaille avec la matrice

```
sage:
A = diagonal_matrix(RealField(24),[k+1 for k in range(N)])
```

sous l'environnement `RealField(24)`. Alors que les valeurs propres devraient bien sûr être $1, ..., \mathtt{N}$, on observe des résultats drastiquement faux si `N` devient grand ; par exemple, pour `N=15` :

```
sage:
A = diagonal_matrix(RealField(24),[k+1 for k in range(15)])
Phi=A.charpoly(); Phi.roots(ComplexField(24))
reponse:
[(-0.509746, 1),
 (0.280585 - 2.28173*I, 1),
 (0.280585 + 2.28173*I, 1),
 (2.42228 - 4.29949*I, 1),
 (2.42228 + 4.29949*I, 1),
 (5.46052 - 5.55291*I, 1),
 (5.46052 + 5.55291*I, 1),
 (8.85051 - 5.83414*I, 1),
 (8.85051 + 5.83414*I, 1),
 (12.2236 - 5.06109*I, 1),
 (12.2236 + 5.06109*I, 1),
 (14.8189 - 3.19445*I, 1),
 (14.8189 + 3.19445*I, 1),
 (16.1985 - 1.11730*I, 1),
 (16.1985 + 1.11730*I, 1)]
```

Le calcul du polynôme caractéristique d'une matrice carrée `A` s'avère, outre sa lourdeur, très instable numériquement (dès lors que des erreurs d'arrondi sont en jeu dans la déclaration des entrées), ce qui fausse drastiquement le calcul du spectre complexe de `A` , même si celui ci s'avère très simple à obtenir directement (ici `A` est diagonale et les valeurs propres complexes sont simplement les éléments diagonaux de la matrice).

Un des buts de ce cours sera de mettre l'accent sur diverses méthodes itératives permettant (entre autres) de calculer le spectre d'une matrice carrée toujours de manière approchée, mais cette fois de manière beaucoup moins instable numériquement que ne l'est cette approche directe *via* le polynôme caractéristique.

Voici ici deux résultats « garde-fou » permettant de localiser (certes parfois grossièrement) les valeurs propres complexes d'une matrice carrée. Le premier est un résultat publié par S. Gerschgorin en 1931 et dont une des preuves se fonde (voir la remarque 1.6 ci-dessous) sur un lemme d'algèbre linéaire de Jacques Hadamard, d'où le fait d'y attacher aussi le nom de Hadamard.

THEORÈME 1.1 (J. Hadamard, S. Gerschgorin [1]). *Soit* `A` *une matrice carrée à coefficients complexes. Chaque valeur propre complexe* λ *de* `A` *se trouve prisonnière dans au moins un disque de Gershgorin attaché à* `A` *, les* `N` *disques de Gerschgorin attachés à* `A` *étant les* `N` *disques du plan complexe :*

$$D_j^{\mathrm{Gersch,A}} := \big\{ z \in \mathbb{C}\,;\, |z - a_{j,j}| \leq \sum_{\substack{k=0 \\ k\neq j}}^{\mathtt{N}-1} |a_{j,k}| \big\}, \qquad j = 0, ..., \mathtt{N}-1.$$

Remarque 1.5. Comme `A` et sa transposée ${}^t\mathtt{A}$ = `A.transpose()` ont mêmes polynômes caractéristiques, donc mêmes spectres complexes, on peut aussi affirmer

1. Ce résultat a été formulé par le mathématicien bielorusse Semyon Aranovich Gershgorin en 1931.

que toute valeur propre de $\mathtt{A}$ se trouve dans au moins l'un des disques de Gerschgorin $D_k^{\mathrm{Gersch},{}^t\mathtt{A}}$, où

$$D_k^{\mathrm{Gersch},{}^t\mathtt{A}} := \big\{ z \in \mathbb{C} \,;\, |z - a_{k,k}| \leq \sum_{\substack{j=0 \\ j\neq k}}^{\mathtt{N}-1} |a_{j,k}| \big\}, \qquad k = 0, ..., \mathtt{N}-1.$$

Démonstration. Soit λ une valeur propre complexe. Il existe donc un vecteur propre $(x_0, ..., x_{\mathtt{N}-1}) \in \mathbb{C}^{\mathtt{N}}$ correspondant à cette valeur propre λ, ce qui signifie $(x_0, ..., x_{\mathtt{N}-1}) \neq 0$ et $\mathtt{A} * x = \lambda\, x$. Soit $j_0 \in \{0, ..., \mathtt{N}-1\}$ tel que

$$|x_{j_0}| = \max_{0\leq k\leq \mathtt{N}-1} |x_k| > 0.$$

Du fait que $\mathtt{A} * x = \lambda\, x$, on a (en prenant les coordonnées d'indice j_0 ce ces deux vecteurs) :

$$(\lambda - a_{j_0,j_0})\, x_{j_0} = \sum_{\substack{k=0 \\ k\neq j_0}}^{\mathtt{N}-1} a_{j_0,k}\, x_k.$$

Du fait de l'inégalité triangulaire, on en déduit

$$|\lambda - a_{j_0,j_0}| \leq \sum_{\substack{k=0 \\ k\neq j_0}}^{\mathtt{N}-1} |x_k|\, |a_{j_0,k}| \leq |x_{j_0}| \sum_{\substack{k=0 \\ k\neq j_0}}^{\mathtt{N}-1} |a_{j_0,k}|.$$

En divisant par $|x_{j_0}| \neq 0$, on en déduit que λ appartient au disque de Gerschgorin (relatif à $\mathtt{A}$) $D_{j_0}^{\mathrm{Gersch},\mathtt{A}}$, donc au moins à l'un des disques de Gerschgorin relatifs à $\mathtt{A}$, ce que l'on voulait montrer. □

Remarque 1.6 (la contribution de J. Hadamard)**.** On verra une autre preuve ultérieurement, basée sur un argument de Jacques Hadamard : dire de λ est valeur propre complexe de $\mathtt{A}$ équivaut à dire que la matrice $\lambda \mathrm{Id}_{\mathtt{N}} - \mathtt{A}$ (considérée comme matrice à entrées complexes) n'est pas inversible comme élément de $\mathscr{M}(\mathbb{C}; \mathtt{N}, \mathtt{N})$. Cette matrice ne saurait donc être « à diagonale dominante », c'est-à-dire telle que

$$\forall\, j \in \{0, ..., \mathtt{N}-1\},\ |\lambda - a_{j,j}| > \sum_{\substack{k=0 \\ k\neq j}}^{\mathtt{N}-1} |a_{j,k}| \ ; \tag{1.13}$$

en effet, on verra plus loin (algorithmes de Jacobi ou de Gauß-Seidel, section 3.2.2) qu'une telle matrice (à diagonale dominante) est automatiquement inversible (l'inverse se calculant d'ailleurs asymptotiquement par une méthode itérative basée sur le théorème du point fixe, voir les cours d'Analyse de S3 et S4). La condition (3.1) est donc en défaut, ce qui prouve aussi que λ est dans l'un au moins des disques de Gerschgorin relatifs à $\mathtt{A}$.

Remarque 1.7 (le cas intéressant où les disques de Gerschgorin relatifs à $\mathtt{A}$ sont deux à deux disjoints)**.** Lorsque les disques de Gerschgorin relatifs à $\mathtt{A}$ sont deux à deux disjoints, on est assuré que le polynôme $\Phi_{\mathtt{A}}$ admet $\mathtt{N}$ valeurs propres distinctes et que chaque disque de Gerschgorin en contient exactement une [1]. Dès

1. On se contentera d'admettre ici ce résultat, conséquence d'un argument de déformation trop délicat à exposer au niveau de ce cours de L3.

lors, on dispose ainsi d'une première information pour « isoler » les valeurs propres dans des « boites » disjointes (ici des disques) pour mieux ensuite pouvoir les calculer. Évidemment l'appartenance à l'union des disques de Gerschgorin relatifs à une matrice carrée `A` ne constitue pas une information suffisamment précise sur la localisation dans le plan complexe des valeurs propres complexes de `A` , il faut ensuite « affiner » par d'autres méthodes, que l'on détaillera par la suite de ce cours.

L'autre résultat concerne la relation entre rayon spectral et choix d'une norme matricielle.

PROPOSITION 1.1 (rayon spectral et normes matricielles). *Soit* $\|\ \|$ *une norme sur* $\mathbb{K}^{\mathtt{N}}$ *et* $\|\ \|$ *la norme matricielle induite sur le* $\mathbb{K}$*-espace vectoriel* $\mathscr{M}(\mathbb{K};\mathtt{N},\mathtt{N})$ *(*$\mathbb{K}=\mathbb{Q},\mathbb{R},\mathbb{C}$*). Pour toute matrice* `A` *de* $\mathscr{M}(\mathbb{K};\mathtt{N},\mathtt{N})$*, on a* $\rho(\mathtt{A})\leq\|\mathtt{A}\|$*. De plus, pour tout* $\epsilon>0$*, il existe une norme* $\|\ \|_{\mathtt{A},\epsilon}$ *sur* $\mathbb{K}^{\mathtt{N}}$ *telle que, pour la norme matricielle induite* $\|\ \|_{\mathtt{A},\epsilon}$ *sur* $\mathscr{M}(\mathbb{K};\mathtt{N},\mathtt{N})$*, on ait :*

$$\|\mathtt{A}\|_{\mathtt{A},\epsilon}\leq\rho(\mathtt{A})+\epsilon\,.$$

DÉMONSTRATION. On prouve d'abord la première assertion (facile). Soit $\|\ \|$ une norme sur $\mathbb{K}^{\mathtt{N}}$ et $\|\ \|$ la norme matricielle correspondante sur le $\mathbb{K}$ espace vectoriel des matrices de taille `[N,N]` à entrées dans $\mathbb{K}$, ce qui signifie :

$$\|\mathtt{A}\|=\sup_{v\in\mathbb{K}^{\mathtt{N}}\setminus\{0\}}\frac{\|\mathtt{A}*v\|}{\|v\|}.$$

Soit λ une valeur propre complexe de `A` telle que $|\lambda|=\rho(A)$ et v un vecteur propre associé. On a $\mathtt{A}*v=\lambda v$, donc $\|\mathtt{A}*v\|=|\lambda|\,\|v\|\leq\|A\|\,\|v\|$. En divisant par $\|v\|$, on trouve $\rho(\mathtt{A})\leq\|\mathtt{A}\|$ comme on le voulait. Pour la seconde assertion, on peut sans restriction supposer $\mathbb{K}=\mathbb{C}$ et on utilise le fait suivant : il existe toujours une base $\{v_{\mathtt{A},0},...,v_{\mathtt{A},\mathtt{N}-1}\}$ de $\mathbb{C}^{\mathtt{N}}$ (dépendant de `A` bien sûr) dans laquelle l'endomorphisme $L_{\mathtt{A}}$: $\mathbb{C}^{\mathtt{N}}\to\mathbb{C}^{\mathtt{N}}$ est représenté par une matrice $\mathtt{R}_{\mathtt{A}}$ triangulaire supérieure (à entrées dans $\mathbb{C}$) [1]. Ceci signifie que

$$\mathtt{A}=\mathtt{P}^{v_{\mathtt{A},0},...,v_{\mathtt{A},\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}}*\mathtt{R}_{\mathtt{A}}*(\mathtt{P}^{v_{\mathtt{A},0},...,v_{\mathtt{A},\mathtt{N}-1}}_{e_0,...,e_{\mathtt{N}-1}})^{-1}.$$

Quitte à effectuer un changement de base dans $\mathbb{C}^{\mathtt{N}}$ (celui transformant la base canonique de $\mathbb{K}^{\mathtt{N}}$ en la base $\{v_{\mathtt{A},0},...,v_{\mathtt{A},\mathtt{N}-1}\}$, on peut se ramener pour prouver la seconde assertion au cas où `A` est une matrice triangulaire supérieure $\mathtt{R}_{\mathtt{A}}$ (à coefficients complexes). Le spectre de `A` et celui de $\mathtt{R}_{\mathtt{A}}$ (constitué des éléments diagonaux de cette matrice triangulaire supérieure) sont les mêmes car les deux matrices `A` et $\mathtt{R}_{\mathtt{A}}$

1. Il s'agit ici d'un résultat établi dans le cours d'Algèbre 2 (S3) et attribué à aux astronomes et mathématiciens (entre autres algébristes) britanniques Arthur Cayley, 1821-1895 (le père de la notion de polynôme caractéristique) et William Hamilton, 1805-1865 : toute matrice à entrées complexes est *trigonalisable*, c'est-à-dire semblable à une matrice triangulaire supérieure ; la preuve de ce résultat est algorithmique et repose sur le théorème fondamental de l'algèbre (tout polynôme à coefficients complexes admet au moins une valeur propre dans $\mathbb{C}$). Il faut pour le coupler avec une procédure algorithmique (récursive), disposer d'un algorithme (on en verra plus loin dans le cours, avec l'algorithme dit « de la puissance ») qui, étant donné une matrice carrée `Aaux` à entrées complexes de taille `[k,k]` (avec `k` $=1,...,$`N`), retourne (au moins) une valeur propre complexe de `Aaux` (en général de module maximum) en même temps qu'un vecteur propre (complexe) associé de `Aaux` associé à cette valeur propre ; la procédure se construit alors récursivement.

sont semblables. Fixons donc $\mathtt{C} > 0$ assez grand et introduisons la matrice diagonale $\mathtt{DC} = \mathtt{diagonal\_matrix}([1, \mathtt{C}, \mathtt{C}^2, ..., \mathtt{C}^{N-1}])$. On remarque que l'on a

$$(\mathtt{DC} * \mathtt{R_A} * \mathtt{DC}^{-1})_{j,k} = \begin{cases} (\mathtt{RA})_{j,k}\mathtt{C}^{j-k} & \text{si} \quad k \geq j \\ 0 \quad \text{si} \quad k < j. \end{cases}$$

Il suffit donc, pour réaliser notre objectif, à savoir disposer d'une norme matricielle $\| \ \|_{\mathtt{R_A},\epsilon}$ telle que

$$\|\mathtt{R_A}\|_{\mathtt{R_A},\epsilon} \leq \rho(\mathtt{R_A}) + \epsilon = \rho(\mathtt{A}) + \epsilon,$$

de choisir la norme $\| \ \|_{\mathtt{R_A},\epsilon}$ sur $\mathbb{C}^N$ de manière à ce que la norme matricielle associée soit la norme définie sur le $\mathbb{C}$-espace des matrices de taille `[N,N]` par

$$\|\mathtt{Matrix}\|_{\mathtt{C_A},\epsilon} = \|\mathtt{DC_A}^{-1} * \mathtt{Matrix} * \mathtt{DC_A}\|_\infty,$$

la constante strictement positive $\mathtt{C_A}$ conditionnant la définition de la matrice $\mathtt{DC_A}$ étant choisie suffisamment grande. La seconde assertion est ainsi prouvée pour la matrice triangulaire supérieure $\mathtt{R_A}$, donc modulo un changement de base pour toute matrice `A` à coefficients complexes, en particulier pour toute matrice `A` à coefficients dans $\mathbb{K}$. □

Pour clôre cette section et souligner l'importance du polynôme caractéristique dans un contexte algébrique général, on rappelle ici le résultat majeur d'Arthur Cayley et William Hamilton :

THEORÈME 1.2 (théorème de Cayley-Hamilton). *Soit* `A` *une matrice carrée à coefficients dans un corps commutatif*[1] $\mathbb{K}$ *et* $\Phi_\mathtt{A}$ *son polynôme caractéristique. On a la relation matricielle* $\Phi_\mathtt{A}(\mathtt{A}) = 0$ *(le second membre figurant la matrice nulle), plus précisément :*

(1.14)

$$\mathtt{A}^{\ N} - \mathtt{A.trace()} * \mathtt{A}^{N-1} + \cdots + (-1)^N * \mathtt{A.determinant()} * \mathtt{identity\_matrix(N)} = \mathtt{zero\_matrix(N,N)}.$$

Plutôt que de donner ici une preuve de ce résultat (voir le cours d'Algèbre 2 en S3), en voici une validation « expérimentale » sous `Sage` pour les matrices de taille $[\mathtt{N}, \mathtt{N}]$ à coefficients dans $\mathbb{Z}$, `taille` désignant la valeur absolue maximale des entrées de la matrice, ces entrées étant générées de manière aléatoire :

```
def VALIDCAYLEY(N,taille):
     R.<X>   = ZZ []; RR.<Y> = MatrixSpace(ZZ,N,N)[]
     M = Matrix([[ZZ.random_element(x=-taille,y=taille)
           for j in range(N)] for k in range (N)])
     polcar  = (M - X*matrix.identity(N)).determinant()
     Lpolcar = polcar.list(); POLCAR = RR(Lpolcar)
     print POLCAR(M) == matrix.zero(N,N)
```

La réponse attendue (quelque soient les valeurs de `N` et de l'entier strictement positif `taille`) se doit d'être `True`, ce qui correspond précisément à la validation du théorème de Cayley-Hamilton pour les matrices carrées à coefficients entiers. On notera la lourdeur (et le temps !) des calculs (de produits de matrices) lorsque `N` devient grand.

1. On pourrait ici se contenter d'un anneau commutatif.

Pour se rendre compte toutefois de l'instabilité des calculs dans le cadre des calculs avec pertes (numériques), on pourra exécuter le code suivant :

```
sage:
def CAYLEYnum(Ndigits,N,taille):
     corpsR  = RealField(Ndigits)
     R.<X>   = corpsR[]; RR.<Y> = MatrixSpace(corpsR,N,N)[]
     M = Matrix([[corpsR.random_element(-taille,taille)
           for j in range(N)] for k in range (N)])
     polcar  = (M - X*matrix.identity(N)).determinant()
     Lpolcar = polcar.list(); POLCAR = RR(Lpolcar)
     return POLCAR(M).norm(2)
```

On travaille cette fois sous l'environnement numérique `RealField(Ndigits)` avec une matrice dont les entrées sont des nombres réels (sous cet environnement) entre `-taille` et `taille`; la réponse devrait être 0 car on demande au code d'afficher la norme euclidienne de la matrice $\Phi_{\mathtt{A}}(\mathtt{A})$ (qui est la matrice nulle d'après le théorème de Cayley-Hamilton). On constate que c'est loin d'être le cas ! Par exemple, avec `Ndigits = 53` et `taille = 100` (pour rectifier le tir, il faut augmenter la précision, ce que l'on fait ensuite en prenant `Ndigits = 500`, mais sans toutefois que cela règle le problème lorsque `taille` devient grand) :

```
sage: CAYLEYnum(53,10,100);
reponse:
11174666424.54184
sage: CAYLEYnum(500,10,100)
1.3697804711519754e-125
sage: CAYLEYnum(500,30,50000)
reponse:
1.0379530810997999e+18
```

1.6. La notion de *conditionnement* d'une matrice

1.6.1. Illustration par l'exemple. Nous avons choisi ici un exemple mettant en lumière les problèmes inhérents au traitement des problèmes d'algèbre linéaire dans le cadre du calcul numérique (ou scientifique). Cet exemple a pour but de sensibiliser dès à présent à une notion relevant strictement du calcul avec pertes et à laquelle il convient de prendre garde très sérieusement lorsque l'on envisage l'algèbre linéaire dans un tel cadre : la notion de *conditionnement*. On déclare sous `Sage` la matrice suivante :

```
sage:
A  = Matrix([[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
```

Comme les entrées sont des entiers, on peut considérer pareille déclaration comme une déclaration relevant du calcul symbolique. Si l'on entend traiter cette matrice comme une matrice dont les entrées sont des nombres réels évalués avec une précision de `N` bits, la déclaration doit être :

```
sage:
Ndigits = [a fixer]
```

```
Anum = Matrix(RealField(Ndigits),
    [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
```

(on peut aussi convenir de remplacer `RealField(N)` par `RDF` si l'on convient de travailler en double précision, comme c'est le cas lorsque l'on se place dans le cadre du calcul scientifique en utilisant `MATLAB` ou `Scilab`). Comme on le vérifie immédiatement (par exemple avec `Sage`), cette matrice carrée `A`, considérée comme une matrice à coefficients dans $\mathbb{Q}$ si on l'envisage sous l'angle du calcul symbolique, est de déterminant 1, son inverse $\mathtt{A}^{-1}$ étant donc égale à la transposée de la matrice de ses cofacteurs [1] :

```
sage:
det(A); A^(-1)
reponse:
1
[ 25 -41  10  -6]
[-41  68 -17  10]
[ 10 -17   5  -3]
[ -6  10  -3   2]
```

Il s'agit ici de calcul symbolique, donc sans pertes. Cependant, si l'on travaille avec une précision arbitraire, ces résultats se trouvent (aux erreurs d'arrondi près) corroborés ; en effet :

```
Ndigits = 20
Anum = Matrix(RealField(Ndigits),
      [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
det(Anum); Anum^(-1)
reponse:
1.0000
[ 25.005 -41.008  10.002 -6.0013]
[-41.008  68.014 -17.004  10.002]
[ 10.002 -17.004  5.0009 -3.0006]
[-6.0013  10.002 -3.0006  2.0003]
```

Comme on le remarque immédiatement :

```
sage:
v = vector([32,23,33,31]); A^(-1)*v
reponse:
(1, 1, 1, 1)
```

ou, si l'on envisage le calcul « avec pertes » en double précision :

```
Anum = Matrix(RDF,
       [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
vnum = vector(RDF,[32,23,33,31]); Anum^(-1)*vnum
```

1. Qui dit calcul de cofacteur dit calcul de déterminant, ce qui pose problème pour des matrices carrées de grande taille ; la formule $\mathtt{A}^{-1} = {}^t[\text{cofacteurs}(\mathtt{A})]$, utilisée ici en petite dimension, n'est pas une formule exploitable pratiquement en grande dimension ; on lui substituera des méthodes d'inversion directe (pivot de Gauß, décomposition `A = LU`, etc.) ou indirectes (Jacobi, Gauß-Seidel, etc.) ultérieurement.

```
reponse:
(1.0, 0.9999999999998863, 1.0000000000000568,
0.9999999999999858)
```

Supposons maintenant que la « mesure » du vecteur `v` soit entachée d'une erreur de mesure aléatoire, ce qui est évidemment le cas dans les problèmes auxquels on se trouve confronté lorsque l'on travaille dans le cadre du calcul scientifique. Pour fixer les idées, supposons que l'erreur de mesure sur `vnum` soit un vecteur dont les quatre coordonnées constituent un échantillon d'une variable aléatoire suivant une loi normale centrée d'écart-type `erreur`. Voici comment une telle erreur peut être générée sous `Sage` :

```
sage:
err    = vector(RDF, 4, {index: normalvariate(0,erreur)
         for index in range(4)})
```

Examinons sous `Sage` l'effet qu'a le fait de perturber additivement préalablement `vnum` (avec une erreur de mesure `err` correspondant à un échantillon d'une variable aléatoire suivant une loi gaussienne centrée d'écart-type $1/100$) sur le calcul $\texttt{A}^{-1}.\texttt{vnum}$ envisagé plus haut :

```
sage :
Anum  = Matrix(RDF,
        [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
vnum  = vector(RDF,[32,23,33,31])
err   = vector(RDF, 4, {index: normalvariate(0,1/100)
         for index in range(4)})
Anum^(-1)*(vnum + err)
reponse:
(1.4756905386292658, 0.20064790796976695,
1.2168014939357192, 0.8708435717682477)
```

On observe[1] une erreur très significative avec le résultat qui, si le second membre `vnum` n'avait pas été perturbé, aurait du être $(1.0, 1.0, 1.0, 1.0)$ (ici presque une erreur relative de 80% sur la seconde coordonnée par exemple). La résolution du système de Cramer $\texttt{Anum} * \texttt{X} = \texttt{vnum}$ s'avère ici totalement sous l'effet d'une erreur de mesure (pourtant petite, ici $\texttt{err} = 1/100$) affectant le second membre `vnum` de ce système. On observe qu'augmenter la précision des calculs (en remplaçant `RDF` par `RealField(N)`) en conservant la même valeur (ici 10^{-2}) pour l'écart-type correspondant à l'erreur additive sur ce second membre `vnum` ne saurait pallier à cette instabilité, bien au contraire ! Par exemple, si $\texttt{N} = 100$ (`RDF` correspondait sensiblement à 53) :

```
sage :
Anum = Matrix(RealField(100),
        [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
vnum = vector(RealField(100),[32,23,33,31])
```

1. Chaque fois que l'on lance cette suite d'instructions, le résultat retourné s'avère différent, du fait que l'erreur de mesure dont est entachée l'évaluation de `vnum` est générée aléatoirement (comme un échantillon de variable gaussienne d'écart type $1/100$). Ceci vaut pour tous les calculs conduits ci-dessous dans cet exemple.

```
err  = vector(RealField(100), 4,
 {index: normalvariate(0,1/100) for index in range(4)})
Anum^(-1)*(vnum + err)
reponse:
(1.435867511599715130865551059O,
0.27508532864306397191719866316,
1.18247482184979711651237610 50,
0.89332380311595012715650265806)
```

Ce n'est qu'en prenant l'écart type `err` correspondant à une erreur de mesure plus petite que l'on peut espérer améliorer la fiabilité numérique des calculs ; mais ceci présuppose être à même d'améliorer la précision des mesures et oblige donc à se que l'on soit tributaire de l'appareillage avec lequel le second membre du système de Cramer à résoudre a été mesuré. Par exemple ici :

```
sage :
Anum  = Matrix(RealField(100),
       [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
vnum  = vector(RealField(100),[32,23,33,31])
err   = vector(RealField(100), 4,
 {index:normalvariate(0,1/1000) for index in range(4)})
Anum^(-1)*(vnum + err)
reponse:
(1.00318393191806323668522141O7,
0.99428380145568891516297954728,
1.00273347143348336016821162 33,
0.99803063698076546978246725840)
```

Au lieu d'envisager une perturbation du second membre du système (qui est ici de Cramer) $\texttt{Anum} * \texttt{X} = \texttt{vnum}$, on peut envisager une perturbation de la matrice `Anum` et supposer que les entrées de cette matrice ne sont connues qu'à une erreur (additive) de mesure près. Pour fixer encore les idées, supposons que l'erreur de mesure sur `Anum` soit une matrice réelle $[4, 4]$ dont les 16 entrées constituent un échantillon d'une variable aléatoire gaussienne centrée d'ecart-type `erreur`. Voici comment une telle erreur matricielle `Err` peut être générée sous `Sage` :

```
sage:
Err = Matrix(RDF,4,4,lambda i,j:normalvariate(0,erreur))
```

On observe ici par exemple l'effet d'une telle perturbation de `Anum` sur le calcul numérique en double précision de $\texttt{Anum}^{-1} * \texttt{vnum}$:

```
sage :
Anum = Matrix(RDF,
        [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
vnum = vector(RDF,[32,23,33,31])
Err  = Matrix(RDF,4,4,lambda i,j:normalvariate(0,1/100))
(Anum+Err)^(-1)*vnum
reponse:
(1.226448149892974, 0.6416874001062922,
```

```
1.0484175315619026, 0.9790277685033786)
```

Ici encore, le fait de remplacer `RDF` par `RealField(N)` avec `N` plus grand que 53 n'améliore pas les choses ; ce n'est qu'en prenant un écart-type de 10^{-4} pour l'erreur de mesure sur les entrées de `Anum` que les choses s'arrangent. Au travers de cet exemple, nous venons de mettre le doigt sur une question ne relevant que du calcul scientifique qui s'avère cruciale en algèbre linéaire : celle de *conditionnement* des matrices. Ce ne sont pas ici les potentialités de la machine qu'il faut incriminer. C'est cette fois une entité mathématique inhérente non à la machine, mais à la matrice `Anum` en jeu ici, entité que l'on définira plus loin comme le *conditionnement* de cette matrice, qui s'avère responsable de pareils phénomènes d'instabilité numérique se produisant lorsque l'on prétend résoudre le système de Cramer $\texttt{Anum} * \texttt{X} = \texttt{vnum}$. Il s'agit là, on le comprend aisément, de problèmes de nature très sérieuse, qui bien sûr sont totalement hors de propos dans le contexte du calcul symbolique (sans pertes).

1.6.2. Approche heuristique du conditionnement. On se contentera pour l'instant de dire qu'intuitivement le *conditionnement* d'une matrice carrée inversible `A` à coefficients réels ou complexes est le produit de la « taille » de cette matrice `A` par la « taille » de son inverse $\texttt{A}^{-1}$. Reste à préciser cette notion de taille et à envisager comment étendre la définition au cadre des matrices rectangulaires (il n'est plus alors question cette fois de parler d'inverse), ce que nous ferons en introduisant la notion de *norme matricielle* introduite à la section 1.4, puis, plus tard, la notion de « décomposition en valeurs singulières » (et de pseudo-inverse) d'une matrice de taille `[M,N]`. Il est clair ici que, nobnobstant le fait que le déterminant de `Anum` vaille 1.0, on obtient sous `Sage` :

```
sage:
Anum = Matrix(RDF,
       [[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
norm(Anum)*norm(Anum^(-1))
reponse:
2984.092701675525
```

(ici `norm(A)=A.norm(2)` figure la norme matricielle euclidienne). Changer d'indicateur (en l'occurrence de norme matricielle) pour « quantifier » la mesure de la « taille » d'une matrice carrée inversible (ou de son celle de son inverse) n'affecte pas le fait qu'ici le conditionnement (à savoir le produit $\|\texttt{A}\| \times \|\texttt{A}^{-1}\|$) se trouve être significativement grand, ce qui explique les phénomènes délicats d'instabilité numérique auxquels on se trouve confronté lors de la résolution numérique d'un système linéaire avec second membre de 4 équations à 4 inconnues dont `Anum` figure la matrice.

1.6.3. Conditionnement d'une matrice carrée inversible de taille `[N,N]` par rapport au choix d'une norme sur $\mathbb{K}^{\texttt{N}}$. Envisageons donc ici la résolution d'un système de Cramer $\texttt{A} * \texttt{X} = \texttt{Y}$ dont nous tolérons de perturber les entrées du fait d'imprécisions sur la mesure des « sorties », à savoir les entrées du second membre `Y`, voire sur les paramètres du système linéaire lui-même, c'est-à-dire les entrées du tableau carré `A`. Supposons que `A` soit de taille `[N,N]` et soit inversible. Les entrées de `A` sont ici prises dans $\mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) et l'on choisit donc pour les quantifier une

norme $\|\ \|$ sur $\mathbb{K}^{\mathtt{N}}$, induisant ainsi une norme matricielle sur le $\mathbb{K}$-espace vectoriel des matrices à entrées dans $\mathbb{K}$ de taille `[N,N]`.

Notons $\mathtt{A}+\Delta\mathtt{A}$ la matrice $\mathtt{A}$ perturbée et $\mathtt{X}+\Delta\mathtt{X}$ la solution du système (que l'on suppose toujours de Cramer, la perturbation étant assez petite pour que l'on ait $\det(\mathtt{A}+\Delta\mathtt{A})\neq 0$) s'exprimant comme

$$(\mathtt{A}+\Delta\mathtt{A})*(\mathtt{X}+\Delta\mathtt{X})=\mathtt{Y} \tag{1.15}$$

(ici, on ne perturbe pas pour l'instant $\mathtt{Y}$). En mettant ensemble les deux relations $A*\mathtt{X}=\mathtt{Y}$ et (1.15), on trouve immédiatement (par différence)

$$\Delta\mathtt{A}*\mathtt{X}+\mathtt{A}*\Delta\mathtt{X}+\Delta\mathtt{A}*\Delta\mathtt{X}=0\,,$$

ce que l'on réécrit

$$\mathtt{A}*\Delta\mathtt{X}=-\Delta\mathtt{A}*(\mathtt{X}+\Delta\mathtt{X})\,. \tag{1.16}$$

On peut transformer (1.16) en

$$\Delta\mathtt{X}=-\mathtt{A}^{-1}*\Delta\mathtt{A}*(\mathtt{X}+\Delta\mathtt{X})$$

et en déduire

$$\|\Delta\mathtt{X}\|\leq\|\mathtt{A}^{-1}\|\times\|\Delta\mathtt{A}\|\times\|\mathtt{X}+\Delta\mathtt{X}\|\,,$$

ce que l'on écrit (un peu artificiellement)

$$\frac{\|\Delta\mathtt{X}\|}{\|\mathtt{X}+\Delta\mathtt{X}\|}\leq\Big(\|\mathtt{A}\|\times\|\mathtt{A}^{-1}\|\Big)\times\frac{\|\Delta\mathtt{A}\|}{\|\mathtt{A}\|}\,,$$

ou encore

$$\frac{\|(\mathtt{X}+\Delta\mathtt{X})-\mathtt{X}\|}{\|\mathtt{X}+\Delta\mathtt{X}\|}\leq\Big(\|\mathtt{A}\|\times\|\mathtt{A}^{-1}\|\Big)\times\frac{\|\Delta\mathtt{A}\|}{\|\mathtt{A}\|}\,.$$

Le membre de gauche

$$\frac{\|(\mathtt{X}+\Delta\mathtt{X})-\mathtt{X}\|}{\|\mathtt{X}+\Delta\mathtt{X}\|}$$

peut s'interpréter comme une « erreur relative » sur la solution $\mathtt{X}$ tandis qu'au membre de droite, on voit apparaître $\|\Delta\mathtt{A}\|/\|\mathtt{A}\|$ qui correspond (en norme) à l'erreur relative commise sur $\mathtt{A}$. La quantité $\|\mathtt{A}\|\times\|\mathtt{A}^{-1}\|$ qui « contrôle » en un certain sens la stabilité de la résolution du système $\mathtt{A}*\mathtt{X}=\mathtt{Y}$ est, on le voit, appelée à jouer un rôle majeur, ce qui nous conduit donc à la définition suivante et nous permet ainsi de valider l'approche heuristique développée au fil de cette section.

Définition 1.5 (conditionnement d'une matrice carrée inversible). Si $\mathtt{A}$ est une matrice carrée inversible de taille `[N,N]` de nombres réels ou complexes, on appelle *conditionnement* de $\mathtt{A}$ (relativement au choix d'une norme $\|\ \|$ sur $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$, suivant le contexte on l'on se place) la quantité $\|\mathtt{A}\|\times\|\mathtt{A}^{-1}\|$ (la norme matricielle étant ici la norme induite par le choix de la norme $\|\ \|$ dans $\mathbb{K}^{\mathtt{N}}$).

Remarque 1.8 (le cas des matrices rectangulaires)**.** La notion de conditionnement d'une matrice rectangulaire de taille `[M,N]` et de rang maximal $\mathtt{r}=$ `A.rank()` $=\min(\mathtt{M},\mathtt{N})>1$ à entrées réelles ou complexes (relativement à la norme matricielle euclidienne `().norm(2)`) sera introduite dans ce cours ultérieurement, une fois que nous disposerons de la notion de « *décomposition en valeurs singulières* » d'une telle matrice : il suffira de considérer la suite des racines carrées $\sigma_0\geq\cdots\geq\sigma_{\mathtt{r}-1}$ des $\mathtt{r}$ valeurs

propres strictement positives de la matrice hermitienne $\mathtt{A}^* * \mathtt{A}$ (symétrique réelle si $\mathtt{A}$ a des entrées réelles), rangées dans l'ordre décroissant, et de définir le conditionnement (euclidien) de $\mathtt{A}$ comme le quotient $\sigma_0/\sigma_{\mathtt{r}-1}$. Calculer le conditionnement euclidien d'une matrice à entrées complexes, de taille `[M,N]` et de rang maximal $\min(\mathtt{M}, \mathtt{N})$ passe donc par la recherche du spectre complexe (en fait il est réel) d'une matrice hermitienne, question que nous traiterons dans ce cours. On retrouve comme cas particulier la définition du conditionnement des matrices carrées inversibles relativement au choix de la norme euclidienne $\| \ \|_2$ sur $\mathbb{K}^{\mathtt{N}}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$).

CHAPITRE 2

Orthogonalité, décomposition QR et conséquences

2.1. Applications bilinéaires sur $\mathbb{K}^N \times \mathbb{K}^N$ et formes quadratiques sur $\mathbb{K}^N$

2.1.1. Définitions. Soit $\mathbb{K}$ (`corps` sous `Sage`) un corps commutatif (éventuellement un environnement numérique tel que `RDF` ou `RealField(Ndigits)` [1]. Une application *bilinéaire* de $\mathbb{K}^N \times \mathbb{K}^N$ dans $\mathbb{K}$ est une application de la forme :

$$\texttt{BILIN : (v,w)} \in \mathbb{K}^N \times \mathbb{K}^N \longmapsto \texttt{w * A * v} \in \mathbb{K}, \tag{2.1}$$

où `A` désigne une matrice de taille `[N,N]` symétrique et à entrées dans $\mathbb{K} =$ `corps`. Nous avons ici supposé que `v` et `w` étaient déclarés comme des vecteurs :

$$\texttt{v} = \texttt{vector (corps,[...])};\ \texttt{w} = \texttt{vector(corps,[...])}$$

tandis que `A` était déclarée comme une matrice

$$\texttt{A} = \texttt{Matrix (corps, [[...],...,[...]])}$$

Le produit (à droite) `A*v` exprime le résultat de l'action sur `v` (considéré comme vecteur-colonne) de l'application linéaire de matrice `A` dans la base canonique de $\mathbb{K}^N$ tandis que le produit (à gauche) `w*A` figure le résultat du produit (cette fois à droite) de la matrice `A` par la matrice-ligne correspondant au vecteur `v`. Si $x_0, ..., x_{N-1}$ désignent les `N` degrés de liberté de $\mathbb{K}^N$ déclarés ici comme `N` variables indépendantes dans $\mathbb{K}$ et `R` le domaine polynomial $\mathbb{K}[x_0, ..., x_{N-1}]$, on associe naturellement à une telle forme bilinéaire l'élément `FQ_A` du domaine polynomial `R` déclaré comme :

```
sage:
N   = A.ncols()
U   = list(var('u_%d' % i) for i in range(N))
T   = corps[U];R=PolynomialRing(corps,N,T.gens(),order='lex')
v   = vector([R.gens()[k] for k in range(N)])
FQ_A  = v*A*v
```

On notera aussi `FQ_A` l'application polynomiale correspondante.

Lorsque $\mathbb{K}$ est un corps de caractéristique différente de 2 ($1 + 1 \neq 0$), se donner un tel polynôme `FQ_A` dans `R` (homogène et de degré 2) revient à se donner la matrice symétrique `A`, donc la forme bilinéaire correspondante. La matrice `A` s'obtient en multipliant par l'inverse de 2 la *matrice Hessienne* de la fonction polynomiale en `N` variables `FQ_A`, c'est-à-dire la matrice

1. Nous laissons de côté pour l'instant le cas des environnements `CDF` ou `ComplexField(Ndigits)` car nous y introduirons plutôt le concept d'application non plus bilinéaire, mais *sesquilinéaire* à la section suivante 2.2 ; on peut envisager dans cette section le cas $\mathbb{K} = \bar{\mathbb{Q}}$, mais à condition toutefois ici que les entrées de la matrice `A` intervenant dans le problème soit à entrées dans $\bar{\mathbb{Q}} \cap \mathbb{R}$.

```
HESS =
Matrix (corps, [[(FQ_A.derivative(R.gens()[k])).derivative(R.gens()[j])
for  k in range(N)] for j in range(N)])
```

L'application

$$\texttt{QUAD} : \texttt{v} \in \mathbb{K}^{\texttt{N}} \longmapsto \texttt{v * A * v} \in \mathbb{K} \tag{2.2}$$

attachée à la forme bilinéaire `BILIN` est dite *forme quadratique* attachée à `BILIN`.

2.1.2. Réduction de Gauß d'une forme quadratique. On suppose que la caractéristique du corps est différente de 2 ; dans la pratique, nos corps seront $\mathbb{Q}, \bar{\mathbb{Q}}$,`RDF` ou `RealField(Ndigits)`. Un résultat fondamental dû à Gauß assure l'existence (ce tout en en donnant un procédé algorithmique de construction, ce qui pour nous est une priorité) d'une matrice de passage `P` (à entrées dans $\mathbb{K}$) inversible (comme élément de $\mathscr{M}(\mathbb{K}; \texttt{N}, \texttt{N})$) et telle que

```
P.transpose()*A*P = diagonal_matrix([a_0,...a_(N-1)])
```

où les $\texttt{a}_\texttt{k}$ (pour $\texttt{k} = 0, ..., \texttt{N} - 1$) sont des éléments de $\mathbb{K}$ constituant une liste `LP`. Le nombre d'éléments non nuls de cette liste `LP` est indépendant de la matrice `P` ainsi construite : c'est le *rang* de la matrice `A` (on dit aussi le *rang de la forme bilinéaire* définie par `A`). Lorsque $\mathbb{K}$ est un sous-corps de $\mathbb{R}$, le couple constitué du nombre d'entrées strictement positives de la liste `LP` et du nombre d'entrées strictement négatives de cette même liste s'appelle la *signature* de la forme bilinéaire (2.1) ou de la forme quadratique correspondante (2.2). Notons qu'il n'y a absolument pas unicité de la matrice `P` ni de la suite des termes diagonaux de la matrice diagonale `P.transpose()*A*P` ainsi obtenue ; cependant l'intérêt de la réduction de Gauß est qu'elle opère sans que l'on ait à s'échapper dans les calculs du corps $\mathbb{K}$ où se trouvent les entrées de la matrice, corps dans lequel on travaille ($\mathbb{K} =$ `corps`).

La méthode de Gauß est fondée sur la formule du trinôme

$$\texttt{A[0,0]}\, u_0^2 + \big(\texttt{A[0,1]}\, u_1 + \dots\big)\, u_0 =$$
$$= \texttt{A[0,0]}^{-1} * \big(\texttt{FQ\_A.derivative(R.gens()[0])}\big)^2 + \texttt{PHI0}\,(u_1, ..., u_{\texttt{N}-1})$$

(pourvu que `A[0,0]` soit non nul), où `PHI0` désigne un polynôme homogène de degré 2 en $u_1, ..., u_N$ qui représente donc une forme quadratique en `N-1` variables et par conséquent une application bilinéaire associée à une certaine matrice symétrique à entrées dans $\mathbb{K}$ de taille `[N-1,N-1]`, c'est-à-dire en dimension cette fois `N-1<N`. Un changement de variable préliminaire

$$\big(u_0, ..., u_{\texttt{N}-1}\big) \leftrightarrow \big(u_0, u_1 + \alpha_1 u_0, ..., u_{\texttt{N}-1} + \alpha_{\texttt{N}-1} u_0\big),$$

où l'on pourra supposer que

```
alpha[k+1] =
KK (ZZ.random_element(x = -tol, y = tol)) for k in range(N-1)
```

(l'entier strictement positif `tol` figurant un seuil maximal de « tolérance » pour la génération aléatoire), éventuellement répété si nécessaire, nous met précisément dans la situation où `A[0,0]` se trouve être une entrée non nulle de la matrice `A`. Ceci rend possible la réalisation d'un code récursif retournant, étant données déclarées les entrées (le corps `corps` de caractéristique distincte de 2, la matrice symétrique `A`, sa taille `N`, enfin la tolérance `tol = tolerance`), une matrice `P` à coefficients dans le

corps `corps` telle que `P.transpose()*A*P` soit une matrice diagonale à entrées dans ce corps. On propose ici ce code récursif `REDUCTIONGAUSS`, tel que nous l'avons rédigé sous l'environnement `Sage` à partir des considérations ci-dessus.

```
def REDUCTIONGAUSS(corps,A,N,tolerance):
    I = identity_matrix(N); I0 = identity_matrix(N-1)
    if N == 1:
        return Matrix([[1]])
    elif A == 0:
        return I
    else:
        AUX = I
        while A[0,0] == 0:
            rdn = [1] + [corps(ZZ.random_element
            (x=-tolerance,y=tolerance)) for k in range(N-1)]
            RDn = [rdn]
            for k in range(N-1):
                RDn = RDn + [[0] + I0[k].list()]
            RDN = Matrix(corps,RDn)
            A   = RDN*A*(RDN.transpose())
            AUX = RDN*AUX
        U   = list(var('u_%d' % i) for i in range(N))
        T   = corps[U]; R = PolynomialRing(corps,N,T.gens(),
                            order ='lex')
        V   = vector([R.gens()[k] for k in range(N)])
        PHI = V*A*V
        p   = PHI.derivative(R.gens()[0])/(2*A[0,0])
        L   = [p.derivative(R.gens()[k]) for k in range(N)]
        PHI0  = PHI - A[0,0]*p^2
        A0=Matrix(corps,[[(PHI0.derivative(R.gens()[k+1])).
        derivative(R.gens()[j+1]) for k in range(N-1)] for j
        in range(N-1)])/2
        P0  = REDUCTIONGAUSS(corps,A0,N-1,tolerance); Q0 = P0^(-1)
        LL  = [L]
        for k in range(N-1):
            LL = LL + [[0] + Q0[k].list()]
        P   = Matrix(corps,LL)
        return AUX.transpose()*P^(-1)
```

2.1.3. Notion de produit scalaire sur $\mathbb{R}^N$ et orthogonalité attachée. Dans cette section, le corps est $\mathbb{R}$ (ou ses versions numériques) mais les calculs exacts pourront être conduits dans $\mathbb{Q}$ (donc de manière exacte) si la matrice `A` de la forme bilinéaire `BILIN` en jeu est à entrées rationnelles (même chose si ces entrées sont algébriques réelles).

Définition 2.1 (produit scalaire sur $\mathbb{R}^N$). Un produit scalaire sur $\mathbb{R}^N$ est une forme bilinéaire `BILIN` sur $\mathbb{R}^N$ de rang `N` et de signature $(N, 0)$, ce qui équivaut à dire

que la forme quadratique `QUAD` associée est définie positive ($\texttt{QUAD}(v,v) \geq 0$ pour tout $v \in \mathbb{R}^{\texttt{N}}$ et $\texttt{QUAD}(v,v) = 0$ si et seulement si $v = 0$). On notera ainsi le produit scalaire :

$$\langle v, v \rangle_{\texttt{A}} := v * \texttt{A} * v,$$

`A` désignant la matrice symétrique réelle attachée à la forme bilinéaire `BILIN` par (2.1), c'est-à-dire représentant cette forme bilinéaire lorsque l'espace $\mathbb{R}^{\texttt{N}}$ est rapporté à sa base canonique. Le produit scalaire dit *canonique* sur $\mathbb{R}^{\texttt{N}}$ est celui qui est représenté lorsque $\mathbb{R}^{\texttt{N}}$ est rapporté à sa base canonique par la matrice identité `matrix.identity(N)` :

$$\langle v, w \rangle = \langle v, w \rangle_{\texttt{matrix.identity(N)}} = \sum_{j=0}^{\texttt{N}-1} x_j \, y_j$$

si $v = (x_0, ..., x_{\texttt{N}-1})$ et $w = (y_0, ..., y_{\texttt{N}-1})$.

À la donnée d'un produit scalaire $\langle \, , \, \rangle_{\texttt{A}}$ sur $\mathbb{R}^{\texttt{N}}$ est attachée une notion d'*orthogonalité* : deux vecteurs v et w de $\mathbb{R}^{\texttt{N}}$ sont *orthogonaux relativement au produit scalaire* $\langle \, , \, \rangle_{\texttt{A}}$ si et seulement si $\langle v, w \rangle_{\texttt{A}} = 0$ (on note ceci aussi $v \perp_{\texttt{A}} w$). Le résultat majeur soutendant dans $\mathbb{R}^{\texttt{N}}$ équipé d'un produit scalaire $\langle \, , \, \rangle_{\texttt{A}}$ la démarche de *l'algorithmique dite Pythagorienne* (en référence à Pythagore [1] est le *théorème de projection orthogonale* :

THEORÈME 2.1 (théorème de projection orthogonale sur une partie convexe et fermée de $\mathbb{R}^{\texttt{N}}$). *Soit Γ un sous-ensemble de $\mathbb{R}^{\texttt{N}}$ à la fois fermé (toute limite d'une suite d'éléments de Γ reste dans Γ) et convexe (tout segment de $\mathbb{R}^{\texttt{N}}$ joignant deux points de Γ reste inclus entièrement dans Γ)* [2]*. Pour tout vecteur $v \in \mathbb{R}^{\texttt{N}}$, il existe un unique point $\mathrm{Pr}_\Gamma[v] \in \Gamma$ tel que*

$$\big\langle v - \mathrm{Pr}_\Gamma[v], v - \mathrm{Pr}_\Gamma[v] \big\rangle_{\texttt{A}} = \min_{w \in \Gamma} \langle v - w, v - w \rangle_{\texttt{A}}.$$

Cet unique point $\mathrm{Pr}_\Gamma[v]$ de Γ est caractérisé par les deux clauses suivantes :

- *d'une part, c'est un point du sous-ensemble Γ ;*
- *d'autre part, on a :*

$$\forall\, w \in \Gamma, \quad \big\langle w - \mathrm{Pr}_\Gamma[v], v - \mathrm{Pr}_\Gamma[v] \big\rangle_{\texttt{A}} \leq 0. \tag{2.3}$$

Ce résultat est illustré par le schéma de la figure 2.1, qu'il faut toujours avoir en tête car c'est sur lui que repose toute la démarche de l'≪ algorithmique Pythagoricienne ≫.

---

1. La formule de Pythagore générale est la formule

$$\langle v + w, v + w \rangle_{\texttt{A}} = \langle v, v \rangle_{\texttt{A}} + \langle w, w \rangle_{\texttt{A}} + 2 \langle v, w \rangle_{\texttt{A}},$$

le terme ≪ correctif ≫ $2 \langle v, w \rangle_{\texttt{A}}$ étant appelé *terme d'interférence* dans cette formule ; lorsque ce terme d'interférence est nul (on dit alors que v et w ne sont pas $\langle \, , \, \rangle_{\texttt{A}}$-corrélés), on retrouve la classique *formule de Pythagore* : le carré de l'hypothénuse d'un triangle rectangle est égal à la somme des carrés des deux autres côtés.

2. Un modèle de tel sous-ensemble Γ est par exemple le sous-ensemble de $\mathbb{R}^{\texttt{N}}$ défini par un jeu (fini) d'≪ inégalités (larges) ou égalités ≫ :

$$\sum_{j=0}^{\texttt{N}-1} \alpha_{\iota,k} \, x_k (\leq \text{ ou } =) \, \gamma_\iota$$

où les nombres γ_ι sont des nombres réels et les vecteurs $(\alpha_{\iota,0}, ..., \alpha_{\iota,\texttt{N}-1})$ des vecteurs de $\mathbb{R}^{\texttt{N}}$. Ces conditions matérialisent ici un jeu de contraintes.

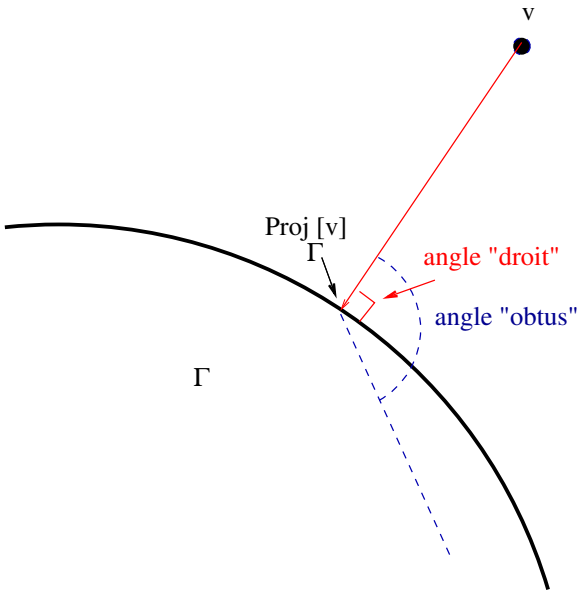

FIGURE 2.1. Projection orthogonale sur une partie convexe fermée

Exemple 2.1 (le cas important où Γ est un sous-espace affine F_a de $\mathbb{R}^{\mathtt{N}}$ contenant le point $a \in \mathbb{R}^{\mathtt{N}}$ et dirigé par le sous-espace vectoriel F de dimension $\mathtt{M} \leq \mathtt{N}$)**.** Dans ce cas particulier, la condition (2.3) s'exprime aussi comme la liste de relations

$$(v - \mathrm{Pr}_{F_a}[v]) \perp_{\mathtt{A}} w \quad \forall\, w \in F.$$

Si F est rapporté à une base $\{v_0, ..., v_{\mathtt{M}-1}\}$ (les v_j étant $\mathtt{M}$ vecteurs de $\mathbb{R}^{\mathtt{N}}$ formant un système libre et engendrant F comme sous-espace vectoriel de $\mathbb{R}^{\mathtt{N}}$), on peut remplacer cette condition (2.3) par le système de m équations

$$\left\langle v - a - \sum_{j=0}^{\mathtt{M}-1} x_j\, v_j, v_k \right\rangle_{\mathtt{A}} = 0 \quad (k = 0, ..., \mathtt{M}-1) \tag{2.4}$$

en disant ensuite que

$$\mathrm{Pr}_{F_a}[v] = a + \sum_{j=0}^{\mathtt{M}-1} x_j\, v_j, \tag{2.5}$$

une fois que $(x_0, ..., x_{\mathtt{M}-1})$ a été calculé comme solution du système de $\mathtt{M}$ équations à $\mathtt{M}$ inconnues (2.4) qui se trouve être un système de Cramer. La matrice de ce système est la matrice réelle symétrique

$$\mathtt{G}_{\mathtt{A}}[v_0, ..., v_{\mathtt{M}-1}] = [\langle v_j, v_k \rangle_{\mathtt{A}}]_{0 \leq j,k \leq \mathtt{M}-1},$$

dite *matrice de Gram* des vecteurs $v_0, ..., v_{\mathtt{M}-1}$ (formant un système libre de $\mathbb{R}^{\mathtt{N}}$) relativement au produit scalaire $\langle v, w \rangle_{\mathtt{A}}$. On a (en théorie du moins et avec ici la syntaxe de `Sage`) :

$$(x_0, ..., x_{\mathtt{M}-1}) = (\mathtt{G}_{\mathtt{A}}[v_0, ..., v_{\mathtt{M}-1}])^{-1} * (v - a).$$

2.2. Applications sesquilinéaires sur $\mathbb{C}^{\mathtt{N}} \times \mathbb{C}^{\mathtt{N}}$ et formes hermitiennes sur $\mathbb{C}^{\mathtt{N}}$

2.2.1. Définitions. Dans le cadre de l'algèbre linéaire sur le corps $\mathbb{C}$ ou un sous-corps $\mathbb{K}$ de ce corps (tel que par exemple $\overline{\mathbb{Q}}$, $\mathbb{Q} + i\,\mathbb{Q}$, $\mathbb{Q}(\alpha, i)$ avec α nombre réel algébrique), une *matrice hermitienne* de taille `[N,N]` est par définition une matrice `A` à entrées complexes (ou dans le sous-corps $\mathbb{K}$ précisé) telle que $\mathtt{a}_{j,k} = \overline{\mathtt{a}_{k,j}}$ pour tout couple d'indices $0 \leq j, k \leq \mathtt{N} - 1$. Il s'agit de l'extension au cadre complexe de la notion de matrice symétrique. Suivant la syntaxe des logiciels de calcul scientifique tels `Scilab` ou `MATLAB`, le fait qu'une matrice `A` (c'est-à-dire un tableau bidimensionnel à entrées des nombres complexes déclarés comme des nombres flottants en double précision) soit hermitienne se traduit par la relation `A = A'` ; sous `Sage`, le fait que `A` soit hermitienne se lit :

```
(A.transpose()).conjugate() == A
```

On peut, dans le cadre de l'environnement `Sage`, envisager les matrices hermitiennes soit à entrées dans `ComplexField(Ndigits)` ou `CDF` (auquel cas le traitement de la matrice s'effectuera dans cet environnement numérique), soit par exemple à entrées dans $\mathbb{K} = \mathbb{Q} + i\,\mathbb{Q} =$ `QQ + I*QQ`, auquel cas ce traitement s'effectuera sur la base du calcul formel conduit dans $\mathbb{Q}$, prenant en compte la relation préalablement déclarée $\mathtt{I}^2 = -1$.

Étant donnée une telle matrice hermitienne `A` (à entrées dans $\mathbb{K} = \mathbb{C}$ ou $\mathbb{K} = \mathbb{Q} + i\,\mathbb{Q}$), on lui associe (la correspondance étant injective) l'application de $\mathbb{K}^{\mathtt{N}} \times \mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{N}}$:

$$\texttt{SESQUI} : \texttt{(v,w)} \in \mathbb{K}^{\mathtt{N}} \times \mathbb{K}^{\mathtt{N}} \longmapsto \texttt{w.conjugate() * A * v} \in \mathbb{K} \tag{2.6}$$

(les conventions d'écriture sont ici celles de l'environnement `Sage`, comme dans la définition des applications bilinéaires (2.1)). Une application de la forme (2.6) est dite *application sesquilinéaire* de $\mathbb{K}^{\mathtt{N}} \times \mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{N}}$. Les applications sesquilinéaires de $\mathbb{K}^{\mathtt{N}} \times \mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{N}}$ sont caractérisées par le fait que

$$\begin{aligned}\texttt{SESQUI}\big(\lambda_0 \mathtt{v}_0 + \lambda_1\,\mathtt{v}_1, \mu_0\,\mathtt{w}_0 + \mu_1\,\mathtt{w}_1\big) &= \lambda_0\overline{\mu_0}\,\texttt{SESQUI}(\mathtt{v}_0, \mathtt{w}_0) + \lambda_1\overline{\mu_1}\,\texttt{SESQUI}(\mathtt{v}_1, \mathtt{w}_1)\\ &\quad + \lambda_0\overline{\mu_1}\,\texttt{SESQUI}(\mathtt{v}_0, \mathtt{w}_1) + \lambda_1\,\overline{\mu_0}\,\texttt{SESQUI}(\mathtt{v}_1, \mathtt{w}_0).\end{aligned}$$

pour tout choix de couples de vecteurs $(\mathtt{v}_0, \mathtt{v}_1)$, $(\mathtt{w}_0, \mathtt{w}_1)$ et de couples d'éléments (λ_0, λ_1), (μ_0, μ_1) de $\mathbb{K}^2$.

Étant donnée une telle application sesquilinéaire de $\mathbb{K}^{\mathtt{N}} \times \mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{N}}$ attachée à une certaine matrice hermitienne `A` à entrées dans $\mathbb{K}$, l'application

$$\texttt{HERM} : \mathtt{v} \in \mathbb{K}^{\mathtt{N}} \longmapsto \texttt{v.conjugate() * A * v} \in \mathbb{K} \cap \mathbb{R} \text{ (ici } \mathbb{Q} \text{ ou } \mathbb{R}) \tag{2.7}$$

est dite *forme hermitienne* attachée à la matrice hermitienne `A`. La formule de polarisation

$$\texttt{SESQUI(v,w)} = \frac{1}{4}\left(\texttt{HERM}(\mathtt{v} + \mathtt{w}) - \texttt{HERM}(\mathtt{v} - \mathtt{w}) + i\,\texttt{HERM}(\mathtt{v} + i\mathtt{w}) - i\,\texttt{HERM}(\mathtt{v} - i\mathtt{w})\right)$$

permet la reconstitution de la forme sesquilinéaire `SESQUI` à partir de la forme hermitienne `HERM` associée.

2.2.2. Réduction de Gauß d'une forme hermitienne. Étant donnée une matrice hermitienne `A` (de taille `[N,N]`) à entrées dans l'un des corps (complexes) $\mathbb{K}$ mentionnés dans cette section, il existe encore une matrice inversible `P` à entrées dans $\mathbb{K}$ et une liste `LP` de nombres réels $[\mathtt{a}_0, ..., \mathtt{a}_{\mathtt{N}-1}]$ appartenant à $\mathbb{K} \cap \mathbb{R}$ tels que, cette fois :

`(P.transpose()).conjugate()*A*P = diagonal_matrix([a_0,...a_(N-1)])`

Le nombre d'éléments non nuls de la liste `LP` est le rang de la matrice hermitienne `A` ; on dit aussi que c'est le rang de la forme hermitienne

$$\mathtt{HERM_A} \ : \mathtt{v} = (u_0, ..., u_{\mathtt{N}-1}) \longmapsto \mathtt{v.conjugate()*A*v}$$

attachée à `A` . La *signature* de cette forme hermitienne est définie comme celle d'une forme quadratique : c'est le couple formé du nombre d'entrées strictement positives de la liste `LP` et du nombre d'entrées strictement négatives de cette même liste. Ni la matrice `P`, ni les entrées de la suite `LP` ne sont uniques, mais par contre rang et signature restent toujours les mêmes indépendamment de la construction de `P`.

Comme pour la réduction de Gauß des formes quadratiques, l'algorithme qui aboutit à une construction de `P` et `LP` (dit *algorithme de réduction de Gauß des formes hermitiennes*) repose sur la remarque suivante. Si `A[0,0]` est non nul, on observe que

$$\begin{aligned} \mathtt{HERM_A(v)} &= \mathtt{A[0,0]}\,|u_0|^2 + \sum_{k=1}^{\mathtt{N}-1} \big(\mathtt{A[0,k]}\,\overline{u_0}\,u_k + \overline{\mathtt{A[0,k]}}\,u_0\,\overline{u_k}\big) + ... \\ &= \mathtt{A[0,0]}\,\Big|u_0 + \sum_{k=1}^{\mathtt{N}-1} \frac{\mathtt{A[0,k]}}{\mathtt{A[0,0]}}\,u_k\Big|^2 + \mathtt{PHI0}((u_1, ..., u_{\mathtt{N}-1})), \end{aligned} \tag{2.8}$$

où `PHI0` est une forme hermitienne en $\mathtt{N}-1$ variables. Si l'on introduit les $2\mathtt{N}$ variables réelles $x_0, y_0, ..., x_{\mathtt{N}-1}, y_{\mathtt{N}-1}$ telles que $u_j = x_j + iy_j$ pour $j = 0, ..., \mathtt{N}-1$ et les opérateurs différentiels

$$\frac{\partial}{\partial u_j} = \frac{1}{2}\Big(\frac{\partial}{\partial x_j} - i\frac{\partial}{\partial y_j}\Big), \quad \frac{\partial}{\partial \bar{u}_j} = \frac{1}{2}\Big(\frac{\partial}{\partial x_j} + i\frac{\partial}{\partial y_j}\Big) \quad (j = 0, ..., \mathtt{N}-1),$$

on remarque que

$$u_0 + \sum_{k=1}^{\mathtt{N}-1} \frac{\mathtt{A[0,k]}}{\mathtt{A[0,0]}}\,u_k = \frac{1}{\mathtt{A[0,0]}}\,\frac{\partial}{\partial \bar{u}_0}\,[\mathtt{HERM_A}((u_0, ..., u_{\mathtt{N}-1}))]$$

et que

$$\mathtt{PHI0} = \mathtt{HERM_A} - \frac{1}{\mathtt{A[0,0]}}\,\frac{\partial[\mathtt{HERM_A}]}{\partial \bar{u}_0}\,\frac{\partial[\mathtt{HERM_A}]}{\partial u_0}.$$

Cette remarque induit une preuve algorithmique (implémentable suivant un code récursif) de la réduction de Gauß des formes hermitiennes. Le corps peut ici être $\mathbb{Q} + i\mathbb{Q}$ (`QQI`) à déclarer préalablement ainsi :

```
RI.<X>= QQ[]; P = X^2+1
QQI.<I> = NumberField(P)
```

si l'on souhaite effectuer des calculs exacts ou `ComplexField(Ndigits)` ou `CDF` si l'on envisage seulement des calculs numériques approchés.

```
def REDUCTIONGAUSSCOMPLEX(corps,A,N,tolerance):
    I    = identity_matrix(N); I0 = identity_matrix(N-1)
    if N == 1:
        return Matrix([[1]])
    elif A == 0:
        return I
    else:
        AUX = I
        while A[0,0] == 0:
            rdn = [1]+[corps(ZZ.random_element
                      (x=-tolerance,y=tolerance))
                      for k in range(N-1)]
            RDn = [rdn]
            for k in range(N-1):
                RDn = RDn + [[0] + I0[k].list()]
            RDN = Matrix(corps,RDn)
            A   = RDN*A*(RDN.transpose())
            AUX = RDN*AUX
        ii   = corps.gens()[0]
        X    = list(var('x_%d' % i) for i in range(N))
        Y    = list(var('y_%d' % i) for i in range(N))
        T    = corps[X+Y]
        R    = PolynomialRing(corps,2*N,T.gens(),order='lex')
        V    = vector([R.gens()[k] + R.gens()[k+N]*ii
                       for k in range(N)])
        Vc   = vector([R.gens()[k] - R.gens()[k+N]*ii
                       for k in range(N)])
        PHI  = Vc*A*V
        p    = (1/2)*(PHI.derivative(R.gens()[0])
             - ii*PHI.derivative(R.gens()[N]))
        pc   = (1/2)*(PHI.derivative(R.gens()[0])
             + ii*PHI.derivative(R.gens()[N]))
        L    = [(pc.derivative(R.gens()[k])
             - ii*pc.derivative(R.gens()[k+N]))/(2*A[0,0])
               for k in range(N)]
        PHI0 = PHI - p*pc/A[0,0]
        M    = [PHI0.derivative(R.gens()[j+1])
             + ii*PHI0.derivative(R.gens()[j+1+N])
                                  for j in range(N-1)]
        A0   = Matrix(corps, [[M[j].derivative(R.gens()[k+1])
             - ii*M[j].derivative(R.gens()[k+1+N])
               for k in range(N-1)] for j in range(N-1)])/4
        P0   = REDUCTIONGAUSSCOMPLEX(corps,A0,N-1,tolerance)
        Q0   = P0^(-1)
        LL   = [L]
```

```
        for k in range(N-1):
            LL = LL + [[0] + Q0[k].list()]
        P   = Matrix(corps,LL)
        return AUX.transpose()*P^(-1)
```

2.2.3. Produit scalaire sur $\mathbb{C}^{\mathtt{N}}$ et orthogonalité attachée. Dans cette section, le corps est $\mathbb{C}$ (ou ses versions numériques) mais les calculs exacts pourront être conduits dans $\mathbb{Q} + i\mathbb{Q}$ (donc de manière exacte) si la matrice `A` de la forme bilinéaire `BILIN` en jeu est à entrées rationnelles (même chose si ces entrées sont algébriques réelles).

DÉFINITION 2.2 (produit scalaire sur $\mathbb{C}^{\mathtt{N}}$). Un produit scalaire sur $\mathbb{C}^{\mathtt{N}}$ est une forme sesquilinéaire `SESQUI` sur $\mathbb{C}^{\mathtt{N}}$ de rang `N` et de signature $(\mathtt{N}, 0)$, ce qui équivaut à dire que la forme hermitienne `HERM` associée est définie positive ($\mathtt{HERM}(v, v) \geq 0$ pour tout $v \in \mathbb{C}^{\mathtt{N}}$ et $\mathtt{QUAD}(v, v) = 0$ si et seulement si $v = 0$). On notera ainsi le produit scalaire :

$$\langle v, v\rangle_{\mathtt{A}} := \mathtt{v.conjugate()*A\ *v},$$

`A` désignant la matrice hermitienne attachée à la forme bilinéaire `SESQUI` par (2.6), c'est-à-dire représentant cette forme bilinéaire lorsque l'espace $\mathbb{C}^{\mathtt{N}}$ est rapporté à sa base canonique. Le produit scalaire dit *canonique* sur $\mathbb{C}^{\mathtt{N}}$ est celui qui est représenté lorsque $\mathbb{C}^{\mathtt{N}}$ est rapporté à sa base canonique par la matrice `matrix.identity(N)` :

$$\langle v, w\rangle = \langle v, w\rangle_{\mathtt{matrix.identity(N)}} = \sum_{j=0}^{\mathtt{N}-1} x_j\, \bar{y}_j$$

si $v = (x_0, ..., x_{\mathtt{N}-1})$ et $w = (y_0, ..., y_{\mathtt{N}-1})$.

À la donnée d'un produit scalaire $\langle\ ,\ \rangle_{\mathtt{A}}$ sur $\mathbb{C}^{\mathtt{N}}$ est attachée la même notion d'orthogonalité que dans le cadre des produits scalaires sur $\mathbb{R}^{\mathtt{N}}$: deux vecteurs v et w de $\mathbb{C}^{\mathtt{N}}$ sont *orthogonaux relativement au produit scalaire* $\langle\ ,\ \rangle_{\mathtt{A}}$ si et seulement si $\langle v, w\rangle_{\mathtt{A}} = 0$ (on note ceci aussi $v \perp_{\mathtt{A}} w$). Le résultat majeur sous tendant dans $\mathbb{C}^{\mathtt{N}}$ équipé d'un produit scalaire $\langle\ ,\ \rangle_{\mathtt{A}}$ la démarche de *l'algorithmique dite Pythagorienne* (en référence à Pythagore [1] est le *théorème de projection orthogonale* :

THEORÈME 2.2 (théorème de projection orthogonale sur une partie convexe et fermée de $\mathbb{C}^{\mathtt{N}}$). *Soit Γ un sous-ensemble de $\mathbb{C}^{\mathtt{N}}$ à la fois fermé (toute limite d'une suite d'éléments de Γ reste dans Γ) et convexe (tout segment de $\mathbb{C}^{\mathtt{N}}$ joignant deux points de Γ reste inclus entièrement dans Γ). Pour tout vecteur $v \in \mathbb{C}^{\mathtt{N}}$, il existe un unique point $\mathrm{Pr}_\Gamma[v] \in \Gamma$ tel que*

$$\big\langle v - \mathrm{Pr}_\Gamma[v], v - \mathrm{Pr}_\Gamma[v]\big\rangle_{\mathtt{A}} = \min_{w\in\Gamma} \langle v - w, v - w\rangle_{\mathtt{A}}.$$

Cet unique point $\mathrm{Pr}_\Gamma[v]$ de Γ est caractérisé par les deux clauses suivantes :

— *d'une part, c'est un point du sous-ensemble Γ ;*

1. La formule de Pythagore générale devient dans ce cadre la formule

$$\langle v + w, v + w\rangle_{\mathtt{A}} = \langle v, v\rangle_{\mathtt{A}} + \langle w, w\rangle_{\mathtt{A}} + 2\,2\,\mathrm{Re}\,(\langle v, w\rangle_{\mathtt{A}}),$$

le terme « correctif » $2\,\mathrm{Re}\,(\langle v, w\rangle_{\mathtt{A}})$ étant appelé *terme d'interférence* dans cette formule ; lorsque ce terme d'interférence est nul (on dit alors que v et w ne sont pas $\langle\ ,\ \rangle_{\mathtt{A}}$-corrélés), on retrouve encore dans ce contexte la classique *formule de Pythagore*.

— *d'autre part, on a :*

$$\forall\, w \in \Gamma, \quad \mathrm{Re}\left\langle w - \mathrm{Pr}_\Gamma[v], v - \mathrm{Pr}_\Gamma[v]\right\rangle_{\mathtt{A}} \leq 0. \tag{2.9}$$

Le schéma de la figure 2.1 reste valable dans ce cadre, à condition que l'on traduise cette fois le fait qu'un angle soit obtus par l'inégalité (2.9).

Exemple 2.2 (le cas important où Γ est un sous-espace affine F_a de $\mathbb{C}^{\mathtt{N}}$ contenant le point $a \in \mathbb{C}^{\mathtt{N}}$ et dirigé par le sous-espace vectoriel F de dimension $\mathtt{M} \leq \mathtt{N}$)**.** Ce qui est valable dans l'exemple 2.1 reste valable à l'inchangé dans ce contexte, à l'exception du fait que la matrice de Gram

$$\mathtt{G}_{\mathtt{A}}[v_0, ..., v_{\mathtt{M}-1}] = [\langle v_j, v_k\rangle_{\mathtt{A}}]_{0\leq j,k\leq \mathtt{M}-1}$$

à inverser pour calculer la projection orthogonale de v sur $a+F$ lorsque $\{v_0, ..., v_{\mathtt{M}-1}\}$ est une base de F est maintenant hermitienne et non plus symétrique réelle.

2.3. Décomposition `Q*R` d'une matrice et variantes

DÉFINITION 2.3. Si `A` est une matrice de taille `[M,N]` à entrées dans $\mathbb{R}$ ou $\mathbb{C}$ (avec $\mathtt{M} \geq \mathtt{N}$), on appelle décomposition `QR` de `A` toute reprśentation de la forme

$$\mathtt{A} = \mathtt{Q} * \mathtt{R}$$

où `R` est une matrice triangulaire supérieure (à entrées réelles si les entrées de `A` le sont, algébriques réelles si les entrées de `A` sont rationnelles) de taille `[M,N]`, ce qui signifie $\mathtt{r}_{j,k} = 0$ si $j > k$, et `Q` est une matrice de taille `[N,N]` dont les vecteurs colonnes forment un système orthonormé relativement au produit scalaire canonique sur $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$ (suivant le cas).

2.3.1. L'algorithme d'orthonormalisation de Gram-Schmidt. Cet algorithme (dans le contexte de $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$) a été vu dans le cours d'Algèbre 2. On en rappelle ici le principe et on le code explicitement sous `Sage`.

Étant donnée une liste de $\mathtt{M} \leq \mathtt{N}$ vecteurs $\mathtt{v}_0$, ...,$\mathtt{v}_{\mathtt{M}-1}$ de $\mathbb{K}^{\mathtt{N}}$ ($\mathbb{K} = \bar{\mathbb{Q}}, \mathbb{R}, \mathbb{C}$, les environnements pouvant être numériques, les coordonnées des vecteurs étant alors déclarés avec une précision de `Ndigits` bits) formant un système libre de $\mathbb{K}^{\mathtt{N}}$ et un produit scalaire $\langle\ ,\ \rangle_{\mathtt{A}}$ sur $\mathbb{K}^{\mathtt{N}}$, on sait construire inductivement un système qui soit orthonormé pour ce produit scalaire dans $\mathbb{K}^{\mathtt{N}}$ et engendre le même sous-espace vectoriel que celui engendré par les vecteurs $\mathtt{v}_k$, $k = 0, ..., \mathtt{M}-1$. La démarche inductive, initiée avec

$$\mathtt{e}_0 = \frac{\mathtt{v}_0}{\|\mathtt{v}_0\|_{2,\mathtt{A}}},$$

(on note $\|\ \|_{2,\mathtt{A}} = \sqrt{\langle\ ,\ \rangle_{\mathtt{A}}}$) se fonde sur la relation de récurrence

$$\mathtt{e}_{k+1} = \frac{\mathtt{v}_{k+1} - \sum_{\ell=0}^{k} \langle \mathtt{v}_{k+1}, \mathtt{e}_\ell\rangle_{\mathtt{A}}\, \mathtt{e}_\ell}{\left\|\mathtt{v}_{k+1} - \sum_{\ell=0}^{k} \langle \mathtt{v}_{k+1}, \mathtt{e}_\ell\rangle_{\mathtt{A}}\, \mathtt{e}_\ell\right\|_{2,\mathtt{A}}} = \frac{\mathtt{v}_{k+1} - \sum_{\ell=0}^{k} \langle \mathtt{v}_{k+1}, \mathtt{e}_\ell\rangle_{\mathtt{A}}\, \mathtt{e}_\ell}{\rho_{k+1}}, \quad k = 0, ..., m-2 \tag{2.10}$$

On note de plus que, si

$$\mathtt{e}_\ell = \sum_{j=0}^{k} a_{j,\ell}\mathtt{v}_j, \quad \ell = 0, ..., k,$$

alors

$$\mathtt{e}_{k+1} = \frac{1}{\rho_{k+1}}\big(\mathtt{v}_{k+1} - \sum_{j=0}^{k}\big(\sum_{\ell=0}^{k}\langle \mathtt{v}_{k+1}, \mathtt{e}_\ell\rangle\, a_{j,\ell}\big)\,\mathtt{v}_j\big), \quad k = 0, ..., m-2. \tag{2.11}$$

Voici cette routine ici, les entrées étant le corps `corps` et la liste `LV` des vecteurs $\mathtt{v}_k$. Le produit scalaire est ici le produit scalaire canonique sur $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$ (suivant le choix du corps `corps`). Ce code retourne la liste `LE` des vecteurs $\mathtt{e}_k$, $k = 0, ..., \mathtt{M}-1$, comme première sortie ; il retourne en seconde sortie la matrice `P` triangulaire supérieure de taille $[\mathtt{M}, \mathtt{M}]$ dont la colonne d'indice k figure la liste des coordonnées du vecteur $\mathtt{e}_k$ exprimé dans la base $\{\mathtt{v}_0, ..., \mathtt{v}_{\mathtt{M}-1}\}$ du $\mathbb{K}$-sous-espace de $\mathbb{K}^{\mathtt{M}}$ engendré par les $\mathtt{v}_k$ (pour $k = 0, ..., \mathtt{M}-1$). La troisième sortie est la matrice (encore triangulaire supérieure) inverse de la matrice obtenue en seconde sortie (calculée inductivement au fil de l'algorithme). La rédaction de ce code s'appuie sur les formules inductives (2.10) (construction itérative de la première sortie) et (2.11) (construction itérative de la seconde sortie).

```
def GRAMSCHMIDT(corps,LV):
    e  = corps(sqrt(LV[0].conjugate()*LV[0]))
    LE = [LV[0]/e]; P  = Matrix(corps,[[1/e]]);
    LPinv = [[e] + [0 for ll in range(len(LV) -1)]]
    for k in range(len(LV)-1):
        C  = [LV[k+1]*(LE[l].conjugate())
              for l in range(k+1)]
        E  = LV[k+1]-sum(C[l]* LE[l] for l in range(k+1))
        e  = corps(sqrt(E.conjugate()*E))
        E  = E/e;LE=LE+[E];p=P*vector(corps,C)
        CC = C + [LV[k+1]*(E.conjugate())]
               + [0 for ll in range(len(LV)-k-2)]
        LPinv = LPinv + [CC]
        LP =[]
        for j in range(k+1):
            LP = LP + [P[j].list()+[-p[j]/e]]
        LP = LP + [[0 for l in range(k+1)] + [1/e]]
        P  = Matrix(corps,LP)
    Pinv = Matrix(corps,LPinv).transpose()
    return LE,P,Pinv
```

Les divisions dans (2.10) et (2.11) rendent cet algorithme parfois numériquement instable : le système libre des vecteurs de la liste `LV` peut être composé en effet de vecteurs « presque colinéaires ». Il ne faut pas perdre de vue en algorithmique que l'orthogonalité n'est pas un concept robuste, mais au contraire fragile ou « friable » : la moindre perturbation de l'un des vecteurs en jeu peut détruire drastiquement l'orthogonalité de deux vecteurs et, par ricochet, provoquer ainsi des erreurs en cascade dans le processus de Gram-Schmidt.

2.3.2. Décomposition `Q*R` *via* l'algorithme de Gram-Schmidt. Soit `A` une matrice de taille `[N,N]` à entrées dans $\mathbb{K} = \bar{\mathbb{Q}}, \mathbb{R}$ ou $\mathbb{C}$ (ce corps sera entré comme `corps` sous `Sage`). On suppose que le rang de `A` est maximal, c'est-à-dire ici égal à `N` .

Les N vecteurs colonnes de la matrice A (éléments de $\mathbb{K}^{\mathtt{N}}$) constituent alors un système libre $\{v_0, ..., v_{\mathtt{N}-1}\}$ de $\mathbb{K}^{\mathtt{N}}$, système que l'on peut orthonormaliser suivant l'algorithme de Gram-Schmidt décrit dans la sous-section précédente. Si l'on considère la matrice Q dont les colonnes figurent les vecteurs $e_0, ..., e_{\mathtt{N}-1}$ et la matrice triangulaire supérieure R telle que

$$\mathtt{r}_{j,k} = \langle v_k, e_j \rangle \quad (0 \leq j, k \leq \mathtt{N} - 1),$$

on obtient la factorisation de la matrice A sous la forme

$$\mathtt{A} = \mathtt{Q*R}.$$

Ici :

— la matrice Q est une matrice dite *orthogonale* (lorsque ses entrées sont réelles) ou *unitaire* (lorsque ses entrées sont complexes), ce qui signifie que la transformation linéaire $L_{\mathtt{Q}}$ est une isométrie de $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$, c'est-à-dire une transformation préservant le produit scalaire canonique, donc la norme euclidienne dans l'espace source (et ici but) $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$; une telle matrice satisfait :

$$\texttt{(Q.transpose()).conjugate()} = \mathtt{Q}^{-1} \ ; \tag{2.12}$$

— la matrice R est une matrice triangulaire supérieure.

Le code suivant réalise cette procédure sous Sage :

```
sage:
def QR_GRAMSCHMIDT(corps,A,N):
    AA = A.transpose(); LV = [AA[j] for j in range(N)]
    G  = GRAMSCHMIDT(corps,LV)
    QL = Matrix(corps,G[0])
    return QL.transpose(),G[2]
```

Remarque 2.1 (le cas des matrices rectangulaires lorsque N < M)**.** Lorsque la matrice est rectangulaire de taille [M,N] avec N < M (mais toujours de rang maximal A.rank() = min(M, N) = N), on peut procéder de manière identique et construire une matrice Q de taille cette fois [M,N] dont les N vecteurs colonnes forment un système orthonormé de $\mathbb{K}^{\mathtt{M}}$ et une matrice triangulaire supérieure R à N lignes et N colonnes, telles que l'on ait encore la factorisation A = Q*R. On peut donc tolérer en entrée dans cet algorithme de factorisation une matrice A de taille [M,N] (à entrées dans corps bien sûr), pourvu que l'on ait N ≤ M. Le code Sage QR_GRAMSCHMIDT proposé ci dessus fonctionne encore dans ce cadre.

2.3.3. Symétries de A. Householder et décomposition Q*R les exploitant. On considère dans cette section le corps $\mathbb{K}$ égal à $\mathbb{R}$ ou $\mathbb{C}$. Le $\mathbb{K}$-espace vectoriel $\mathbb{K}^{\mathtt{N}}$ sera équipé de son produit scalaire canonique $\langle\ ,\ \rangle$.

Étant donné un vecteur v de $\mathbb{K}^{\mathtt{N}}$, supposé non nul, on peut lui attacher de manière naturelle une application linéaire orthogonale dans le cadre réel, unitaire dans le cadre complexe, que l'on notera $L_{\mathtt{Q_v}}$.

DÉFINITION 2.4 (symétrie d'Askon Householder [1]). Étant donné un vecteur non nul de l'espace euclidien ou hermitien $(\mathbb{K}, \langle\ ,\ \rangle)$, on appelle symétrie (orthogonale) de

1. Alston Scott Householder, 1904-1993, est un mathématicien appliqué américain ; ses travaux sont relatifs à l'analyse numérique (ce qui nous préoccupe dans ce cours) et aux mathématiques appliquées à la biologie.

Householder $L_{\mathtt{Q_v}}$ associée au vecteur v l'application linéaire qui à un vecteur w de $\mathbb{K}^{\mathtt{N}}$ associe le symétrique (orthogonalemement) de v par rapport au sous-espace vectoriel (de dimension $\mathtt{N}-1$) $\mathtt{v}^{\perp}$ constitué des vecteurs orthogonaux à v dans $\mathbb{K}^{\mathtt{N}}$. La matrice de $L_{\mathtt{Q_v}}$ lorsque $\mathbb{K}^{\mathtt{N}}$ est rapporté à sa base canonique est donc :

(2.13)

$$\mathtt{Q_v} = \mathtt{matrix.identity(N)} - 2 * \mathtt{V.transpose()} * \mathtt{V.conjugate()}/(\mathtt{norm(v)})^2.$$

(ici V est la matrice-ligne dont les entrées sont les coordonnées de v et norm(v) désigne la norme euclidienne de v, c'est-à-dire la racine carrée de v.congugate()*v si v est déclaré comme un vecteur sous Sage). Il s'agit d'une matrice orthogonale (unitaire dans le cas complexe) qui a la particularité de vérifier $\mathtt{Q_v}^2 = \mathtt{matrix.identity(N)}$ (et par conséquent $\mathtt{Q_v}^* = \mathtt{Q_v}$) puisqu'il s'agit de la matrice d'une symétrie orthogonale (par rapport à l'orthogonal $\mathtt{v}^{\perp}$ de v).

Étant donné un vecteur arbitraire w de $\mathbb{K}^{\mathtt{N}}$, il est toujours possible de réaliser une symétrie de Householder (associée à un vecteur non nul v à choisir convenablement en fonction de w) qui transforme le vecteur w en un vecteur colinéaire au premier vecteur e_1 de la base canonique de $\mathbb{K}^{\mathtt{N}}$.

Dans le cas $\mathbb{K} = \mathbb{R}$, on distingue suivant que w est orthogonal à e_1 ou non :

— dans le premier cas (w[0]=0), on prend v=w ;

— dans le second cas (w[0] $\neq 0$), il suffit de poser (par exemple [1])

$$\mathtt{v} = \mathtt{w} + \|\mathtt{w}\|\, e_1$$

lorsque ce vecteur est non nul [2].

On vérifie bien dans les deux cas que l'on a

$$\mathtt{Q_v}(\mathtt{w}) = -\|\mathtt{w}\|\, e_1.$$

Il suffit pour voir cela de faire un dessin, ce que nous avons fait sur la figure 2.2 suivante.

Dans le cas complexe ($\mathbb{K} = \mathbb{C}$), on procède de manière identique. On distingue encore suivant que w est orthogonal à e_1 ou non :

— dans le premier cas (w[0]=0), on prend v=w ;

— dans le second cas (w[0] $\neq 0$), il suffit de poser (par exemple [3])

$$\mathtt{v} = \mathtt{w} + e^{i\theta}\|\mathtt{w}\|\, e_1$$

(où θ est tel que $\mathtt{w[0]} (\neq 0) = e^{i\theta}|\mathtt{w[0]}|$) lorsque ce vecteur est non nul [4].

On vérifie encore que dans les deux cas on a toujours

$$\mathtt{Q_v}(\mathtt{w}) = -\|\mathtt{w}\|\, e_1.$$

---

1. On pourrait aussi choisir $\mathtt{v} = \mathtt{w} - \|\mathtt{w}\|\, e_1$ mais on privilégie le choix ci-dessus pour des raisons de stabilité.

2. Si ce vecteur est nul, on convient de choisir pour $L_{\mathtt{Q_v}}$ l'identité, ce qui revient à ne rien faire.

3. On pourrait aussi choisir $\mathtt{v} = \mathtt{w} + \lambda\|\mathtt{w}\|\, e_1$ avec un nombre complexe λ de module 1 arbitraire mais on privilégie le choix ci-dessous pour des raisons de stabilité.

4. Si ce vecteur est nul, on convient encore de choisir pour $L_{\mathtt{Q_v}}$ l'identité, ce qui revient à ne rien faire.

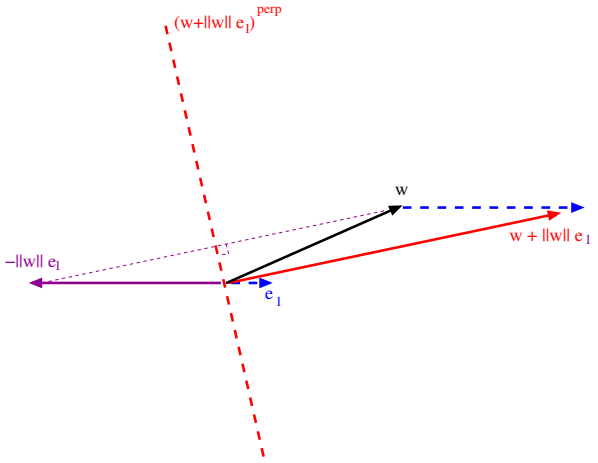

FIGURE 2.2. Symétrie orthogonale de Householder

La réalisation de symétries orthogonales de Householder génère une procédure algorithmique pour décomposer une matrice carrée `A` de taille `[N,N]` sous la forme `A=QR`. Voici le principe de cet algorithme (qui s'implémentera ensuite récursivement) :

- on considère le premier vecteur colonne `w=A[:,1]` de la matrice `A` et l'on réalise la symétrie de Householder $L_{\mathtt{Q_v}}$ transformant `w` en le vecteur $-\|\mathtt{w}\|\, e_1$;
- on observe que $\mathtt{Q_v}*(\mathtt{Q_v}*\mathtt{A})$ est une factorisation de `A` mais que l'on a gagné avec $\mathtt{Q_v}*\mathtt{A}$ à la place de `A` le fait que le premier vecteur colonne de cette nouvelle matrice $\mathtt{Anew} = \mathtt{Q_v}*\mathtt{A}$ est le vecteur $(-\|\mathtt{w}\|, 0, ..., 0)$;
- la matrice `Anew` se présente donc comme une matrice
$$\begin{bmatrix} -\|\mathtt{w}\| & \mathtt{ligne(1,N-1)} \\ \mathtt{zeros(N-1,1)} & \mathtt{Anewbloc(N-1,N-1)} \end{bmatrix} ;$$
- on peut traiter la matrice `Anewbloc` (de taille cette fois `[N-1,N-1]`, d'où la possibilité de récursivité) comme on a traité la matrice `A` au premier cran et construire ainsi (de manière récursive) une matrice $\mathtt{Q_{N-1}}$ de taille `[N-1,N-1]`, orthogonale réelle ou unitaire, ainsi qu'une matrice $\mathtt{R_{N-1}}$ triangulaire supérieure et de carré `matrix.identity(N-1,N-1)`, telles que l'on ait
$$\mathtt{Anewbloc} = \mathtt{Q_{N-1}} * \mathtt{R_{N-1}}$$
(appel à la récursivité) ;
- on complète la matrice $\mathtt{Q_{N-1}}$ en une matrice de taille `[N,N]`, $\mathtt{Q_N}$, elle aussi orthogonale (ou unitaire dans le cas complexe) et de carré `matrix.identity(N)`, ainsi :
$$\mathtt{Q_N} = \begin{bmatrix} 1 & \mathtt{zeros(1,N-1)} \\ \mathtt{zeros(N-1,1)} & \mathtt{Q_{N-1}} \end{bmatrix}$$
La décomposition
$$\mathtt{A} = (\mathtt{Q_v} * \mathtt{Q_N}) * \left(\mathtt{Q_N} * \begin{bmatrix} -\|\mathtt{w}\| & \mathtt{ligne(1,N-1)} \\ \mathtt{zeros(N-1,1)} & \mathtt{Anewbloc(N-1,N-1)} \end{bmatrix}\right)$$

donne la décomposition `A=Q*R` voulue.

Voici ce code sous `Sage`, rédigé ici suivant une procédure directe (non récursive comme dans le synopsis ci-dessus, mais la démarche algorithmique est la même). Ceci est la version « numérique ». Le corps `corps` doit être ici `RDF` ou `CDF` (double précision), ou bien `RealField(Ndigits)` ou `ComplexField(Ndigits)` (précision arbitraire de `Ndigits` bits après la virgule en binaire).

```
def DecompQR(corps,A,N):
    if corps == CDF:
        corpsR = RDF;
    else:
        corpsR = corps.base_ring()
    Anew = A ; Qnew = identity_matrix(N)
#lancement de la boucle de calculs
    for k in range(N):
        a = Anew.matrix_from_rows_and_columns
           ([l+k for l in range(N-k)],[k])
#recherche de la symetrie de Householder
        aa = a.transpose(); x = aa[0]
        if x.norm(2) <= corpsR(1).ulp():
            v=zero_vector(N-k)
        else:
            if abs(x[0]) <= corpsR(1).ulp():
                rho = 0
            else:
                rho = x[0]/abs(x[0])
            normx = x.norm(2)
            vaux  = normx*rho*vector([1]
                    + [0 for l in range(N-k-1)]) + x
            if vaux.norm(2) <= corpsR(1).ulp():
                v=zero_vector(N-k)
            else:
                v = vaux/(vaux.norm(2))
#construction de la matrice de cette symetrie
        V=Matrix(corps, [v.list()])
        H=identity_matrix(N-k)-2*V.transpose()*V.conjugate()
#realisation de la factorisation (principe recursif deguise)
        LP1 = [[0 for l in range(j)]+[1]
              +[0 for l in range(N-j-1)] for j in range(k)]
        LP2 = [[0 for l in range(k)]
              + H[j].list() for j in range(N-k)]
        P   = Matrix(corps,LP1+LP2)
        Qnew   = Qnew*P; Anew = P*Anew
    return Qnew, Anew
```

Pour disposer d'une version exacte (pour travailler sans pertes avec une matrice `A` à entrées algébriques et prendre pour corps `QQbar` $= \bar{\mathbb{Q}}$ car il y a des calculs de norme,

donc des prises de racines carrées dans l'algorithme), il suffit de supprimer la première boucle `if ... else ...` du code ci-dessus et de prendre `corps = QQbar`. La matrice `A` doit alors être à entrées dans ce corps, ce peut être par exemple une matrice à entrées dans $\mathbb{Q}$ ou dans $\mathbb{Q} \otimes i\mathbb{Q}$.

2.3.4. Factorisation `A=Q*H*Q`* de K. Hessenberg d'une matrice carrée réelle ou complexe ; le cas des matrices symétriques réelles ou hermitiennes complexes. La démarche de factorisation d'une matrice carrée réelle ou complexe `A` sous la forme `A=Q*R` en introduisant de manière successive des symétries orthogonales de Householder que l'on compose (voir la sous-section 2.3.3) peut être pensée différemment. Rappelons que l'on avait proposé dans cette section une factorisation de `A` de la forme

$$\mathtt{A} = \left(\mathtt{Q}_{\mathtt{v}_1} * \cdots * \mathtt{Q}_{\mathtt{v}_N}\right) * \left(\mathtt{Q}_{\mathtt{v}_N} * \cdots * \mathtt{Q}_{\mathtt{v}_1} * \mathtt{A}\right) = \mathtt{Q} * \mathtt{R}.$$

On aurait aussi pu introduire la matrice

$$\mathtt{Anew} = \left(\mathtt{Q}_{\mathtt{v}_N} * \cdots * \mathtt{Q}_{\mathtt{v}_1}\right) * \mathtt{A} * \left(\mathtt{Q}_{\mathtt{v}_1} * \cdots * \mathtt{Q}_{\mathtt{v}_N}\right) = \widetilde{\mathtt{Q}} * \mathtt{A} * \widetilde{\mathtt{Q}}^* \quad (\text{avec } \widetilde{\mathtt{Q}} = \mathtt{Q}^*). \tag{2.14}$$

Puisque les matrices $\mathtt{Q}_{\mathtt{v}_j}$, `j=1,...,N`, sont toutes orthogonales réelles (ou unitaires dans le cas complexe), on a $\mathtt{Q}^* = \mathtt{Q}^{-1}$ et la matrice (2.14) représente l'application linéaire $L_{\mathtt{A}}$ non plus cette fois dans la base canonique de $\mathbb{R}^{\mathtt{N}}$ ou $\mathbb{C}^{\mathtt{N}}$, mais dans une nouvelle base, d'ailleurs orthonormée, celle dont les vecteurs sont précisément les vecteurs colonnes de la matrice `Q` (voir la formule de changement de base (1.2)) : on a en effet $\mathtt{Anew} = \mathtt{Q}^{-1} * \mathtt{A} * \mathtt{Q}$, ce conformément à cette formule de changement de base.

Le code réalisant cette démarche sous `Sage` est construit sur le modèle du code précédent. Le voici ici, dans sa version « numérique ». Le corps `corps` doit être ici `RDF` ou `CDF` (double précision), ou bien `RealField(Ndigits)` ou `ComplexField(Ndigits)` (précision arbitraire de `Ndigits` bits après la virgule en binaire). Il retourne en première sortie la matrice `Anew`, en seconde sortie la matrice orthogonale réelle ou unitaire complexe (suivant que `A` soit à entrées dans $\mathbb{R}$ ou dans $\mathbb{C}$) telle que $\mathtt{Anew} = \mathtt{Q} * \mathtt{A} * \mathtt{Q}^*$.

```
def DecompQRHessenberg(corps,A,N):
    if corps == CDF:
        corpsR = RDF;
    else:
        corpsR = corps.base_ring()
    Anew = A; Qnew = identity_matrix(N)
    for k in range(N):
        a = Anew.matrix_from_rows_and_columns
           ([l+k for l in range(N-k)],[k])
        aa = a.transpose(); x = aa[0]
        if x.norm(2) <= corpsR(1).ulp():
            v=zero_vector(N-k)
        else:
            normx=x.norm(2)
            if abs(x[0]) <= corpsR(1).ulp():
                rho=0
```

```
            else:
                rho = x[0]/abs(x[0])
            vaux  = normx*rho*vector([1]
                    + [0 for l in range(N-k-1)]) + x
            if vaux.norm(2) <= corpsR(1).ulp():
                v=zero_vector(N-k)
            else:
                v = vaux/(vaux.norm(2))
        V=Matrix(corps, [v.list()])
        H=identity_matrix(N-k)-2*(V.transpose())*(V.conjugate())
        LP1 = [[0 for l in range(j)]+[1]
              +[0 for l in range(N-j-1)] for j in range(k)]
        LP2 = [[0 for l in range(k)]
              + H[j].list() for j in range(N-k)]
        P   = Matrix(corps,LP1+LP2);
        Anew = P*Anew*((P.transpose()).conjugate())
        Qnew = P*Qnew
    return Anew, Qnew
```

Ici encore, si l'on travaille dans le cadre exact (`corps` = `QQbar`, le corps $\bar{\mathbb{Q}}$ des nombres algébriques), il faut omettre la première boucle `if ... else ...` du code.

Cette démarche constitue une brique essentielle de l'algorithme (dit `QR`) de réduction des matrices orthogonales réelles (ou unitaires) proposé par le mathématicien et ingénieur allemand Karl Hessenberg, 1904-1959, le mathématicien britannique John Francis, 1934 - , ainsi que la mathématicienne soviétique Vera Kublanovskaya. Il s'agit d'un des algorithmes majeurs du siècle dernier, proposé en 1961. L'algorithme de Francis-Kublonovskaya permet en fait le calcul du spectre complexe d'une matrice carrée ainsi (c'est le cas par exemple lorsque cette matrice est symétrique réelle ou hermitienne complexe) que le calcul d'une base constituée de vecteurs propres dans laquelle cette matrice est diagonalisable.

À première vue toutefois, comme on s'en rend compte immédiatement en faisant quelques tests, l'algorithme `A` $\leftarrow$ `Q*A*Q`* semble ne rien apporter : le gain qui à chaque cran de la factorisation `QR` « à la Householder », qui consistait à faire apparaitre à chaque cran un zéro de plus dans la première colonne des blocs, se trouve perdu lorsque l'on opère la multiplication $\widetilde{\mathtt{Q}}_{\mathtt{v}} * \mathtt{Aold} * \widetilde{\mathtt{Q}}_{\mathtt{v}}^*$ pour passer d'une matrice `Aold` à une nouvelle matrice `Anew`. Il semble que l'on ne gagne rien car on ne « crée pas de zéros » !

Nous allons procéder donc différemment pour construire, en modifiant (tout en en conservant l'idée) le code `DecompHessenberg`, une matrice `[N,N]` notée `H` (comme « Hessenberg ») telle que `A=Q*H*Q`* avec `Q` orthogonale réelle ou complexe unitaire et que de plus `H` ne soit certes plus triangulaire supérieure comme dans l'algorithme `QR`, mais ait au moins toutes ses entrées $a_{j,k}$ nulles dès que $j \geq k+1$, c'est-à-dire soit une matrice dont toutes les entrées figurant en dessous de la sous-diagonale soient nulles. Dans le cas particulier où `A` sera symétrique réelle ou hermitienne complexe,

on obtiendra même ainsi une matrice *tridiagonale*, c'est-à-dire une matrice dans laquelle seules la diagonale, la sur-diagonale et la sous-diagonale sont non nulles. Cette procédure sera dite *décomposition* $\mathtt{A} = \mathtt{Q*H*Q}^*$ *de Karl Hessenberg*.

Une fois que nous disposerons d'une telle matrice (dite *de Hessenberg*) `H` telle que $\mathtt{A} = \mathtt{Q*H*Q}^*$ et que `H` ait toutes ses diagonales nulles en dessous de la sous-diagonale (voire même tridiagonale lorsque `A` est symétrique réelle ou hermitienne complexe), nous serons à même (ce sera l'objet de la sous-section suivante) de lancer cette fois avec succès (en l'itérant) l'algorithme $\mathtt{Q*A*Q}^*$ de John Francis et Vera Kublonovskaya.

Voici la démarche de Karl Hessenberg, visant à construire, étant donnée une matrice de taille `[N,N] A`, une matrice orthogonale réelle (dans le cadre réel) ou unitaire complexe (dans le cadre complexe) `Q` telle que $\mathtt{H=Q}^*\mathtt{*A*Q}$ soit une matrice dont les entrées en dessous de la sous-diagonale soient toutes nulles. Dans le cas particulier où la matrice `A` est symétrique réelle (dans le cadre réel) ou hermitienne complexe (dans le cadre complexe), il en sera de même pour la matrice `H` puisque l'on sait que $\mathtt{H}^* = \mathtt{Q}^* * \mathtt{A}^* * \mathtt{Q}$ (la prise d'adjoint d'un produit de matrices carrées de même taille se fait à l'envers et est une opération involutive) et que $\mathtt{A}^* = \mathtt{A}$ dans ce cas particulier ; par conséquent, la réduite de Hessenberg `H` obtenue *via* la méthode décrite ci-dessous lorsque la matrice de départ `A` est symétrique réelle ou hermitienne complexe sera une matrice tridiagonale (par symétrie ou hermitiannité, toutes ses entrées au dessus de la sur-diagonale seront nulles, comme le sont par construction toutes ses entrées en dessous de la sous-diagonale).

— On part de la matrice `A` que l'on écrit

$$\mathtt{A} = \begin{bmatrix} a_{0,0} & \mathtt{L(1,N-1)} \\ \mathtt{C(N-1,1)} & \mathtt{Abloc(N-1,N-1)} \end{bmatrix}$$

(`L` pour « matrice ligne » et `C` pour « matrice colonne ») et on note cette fois `w` le vecteur de $\mathbb{K}^{\mathtt{N}-1}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) dont les entrées sont celles de la matrice colonne `C(N-1,1)`. On détermine un vecteur $\mathtt{v}_1$ (cette fois dans $\mathbb{K}^{\mathtt{N}-1}$ et non plus dans $\mathbb{K}^{\mathtt{N}}$ comme c'était le cas dans la procédure décrite à la sous-section 2.3.3) tel que la symétrie de Householder correspondante (dans $\mathbb{K}^{\mathtt{N}-1}$) envoie ce vecteur `w` sur $-\|\mathtt{w}\|\, e_1^{[\mathtt{N}-1]}$, où $e_1^{[\mathtt{N}-1]}$ désigne cette fois le premier vecteur de la base canonique de $\mathbb{K}^{\mathtt{N}-1}$; on note $\mathtt{Q}_{\mathtt{v}_1}^{[\mathtt{N}-1]}$ la matrice de taille `[N-1,N-1]` représentant cette symétrie de Householder dans la base canonique de $\mathbb{K}^{\mathtt{N}-1}$.

— On forme la matrice

$$\widetilde{\mathtt{Q}}_{\mathtt{v}_1} = \begin{bmatrix} 1 & \mathtt{zeros(1,N-1)} \\ \mathtt{zeros(N-1,1)} & \mathtt{Q}_{\mathtt{v}_1}^{[\mathtt{N}-1]} \end{bmatrix},$$

puis on construit la matrice $\widetilde{\mathtt{Q}}_{\mathtt{v}_1} * \mathtt{A} * \widetilde{\mathtt{Q}}_{\mathtt{v}_1}$. Cette matrice se présente sous la forme

$$\widetilde{\mathtt{Q}}_{\mathtt{v}_1} * \mathtt{A} * \widetilde{\mathtt{Q}}_{\mathtt{v}_1} = \begin{bmatrix} a_{0,0} & \mathtt{L[0,1]} & \mathtt{[L[0,2],...,L[0,N-1]]} \\ -\|\mathtt{w}\| & \beta_{1,1}^{[\mathtt{1}]} & \mathtt{L}_{\mathtt{1}}^{[\mathtt{1}]}\mathtt{(1,N-2)} \\ \mathtt{zeros(N-2,1)} & \beta_{\mathtt{N}-1,1}^{[\mathtt{1}]} & \mathtt{L}_{\mathtt{N}-1}^{[\mathtt{1}]}\mathtt{(1,N-2)} \end{bmatrix}.$$

On a donc gagné des zéros sous la sous-diagonale dans la première colonne (la première ligne est restée la première ligne de la matrice `A` car elle n'a aucunement été touchée par cette procédure).

— On réitère l'opération cette fois en dimension $\mathtt{N}-1$ avec la matrice bloc

$$\begin{bmatrix} \beta_{1,1}^{[1]} & \mathtt{L}_1^{[1]}\mathtt{(1,N-2)} \\ \beta_{\mathtt{N}-1,1}^{[1]} & \mathtt{L}_{\mathtt{N}-1}^{[1]}\mathtt{(1,N-2)} \end{bmatrix}.$$

On construit ainsi une matrice $\mathtt{Q}_{\mathtt{v}_2}^{[2]}$ (de symétrie de Householder dans $\mathbb{K}^{\mathtt{N}-2}$) que l'on complète en une matrice de $\mathtt{Q}_{\mathtt{v}_2}$ ainsi :

$$\widetilde{\mathtt{Q}}_{\mathtt{v}_2} = \begin{bmatrix} \mathtt{matrix.identity(2)} & \mathtt{zeros(1,N-2)} \\ \mathtt{zeros(N-2,2)} & \mathtt{Q}_{\mathtt{v}_2}^{[2]} \end{bmatrix}.$$

On forme la matrice

$$\widetilde{\mathtt{Q}}_{\mathtt{v}_2} * (\widetilde{\mathtt{Q}}_{\mathtt{v}_1} * \mathtt{A} * \widetilde{\mathtt{Q}}_{\mathtt{v}_1}) * \widetilde{\mathtt{Q}}_{\mathtt{v}_2}$$

et ainsi de suite... jusquà obtenir une matrice H (dite *réduite de Hessenberg* de A) dont toutes les entrées sous la sous-diagonale sont nulles (comme on l'espérait).

On observe au final que par construction $\mathtt{H} = \mathtt{Q}^* * \mathtt{A}*\mathtt{Q}$ pour une certaine matrice orthogonale réelle ou unitaire complexe Q, ce qui s'exprime aussi $\mathtt{A} = \mathtt{Q} * \mathtt{H} * \mathtt{Q}^*$. Cette factorisation est dite *factorisation* $\mathtt{Q} * \mathtt{H} * \mathtt{Q}^*$ *de Hessenberg* de la matrice carrée réelle ou complexe A.

Voici ici le code rédigé sous Sage réalisant cette opération (il retourne H en première sortie et Q en seconde sortie). Le corps corps doit être ici RDF ou CDF (double précision), ou bien RealField(Ndigits) ou ComplexField(Ndigits) (précision arbitraire de Ndigits bits après la virgule en binaire).

```
def HESSENBERG(corps,A,N):
    if corps == CDF:
        corpsR = RDF;
    else:
        corpsR = corps.base_ring()
    Anew = A ; Q = identity_matrix(N)
    for k in range(N-2):
        a = Anew.matrix_from_rows_and_columns
           ([l+k+1 for l in range(N-k-1)],[k])
        aa = a.transpose(); x = aa[0]
        if x.norm(2) <= corpsR(1).ulp():
            v = zero_vector(N-k-1)
        else:
            normx = x.norm(2)
            if abs(x[0]) <= corpsR(1).ulp():
                rho = 0
            else:
                rho = x[0]/abs(x[0])
            vaux  = normx*rho*vector([1]
                    + [0 for l in range(N-k-2)]) + x
            if vaux.norm(2) <= corpsR(1).ulp():
                v = zero_vector(N-k-1)
```

```
            else:
                v = vaux/(vaux.norm(2))
        V = Matrix(corps, [v.list()])
        H=identity_matrix(N-k-1)-2*(V.transpose())*(V.conjugate())
        LP1 = [[0 for l in range(j)]+[1]+[0 for l in range(N-j-1)]
              for j in range(k+1)]
        LP2 = [[0 for l in range(k+1)] + H[j].list()
              for j in range(N-k-1)]
        P   = Matrix(corps,LP1+LP2)
        Q   = P*Q
        Anew = P*Anew*((P.transpose()).conjugate())
    return Anew,(Q.transpose()).conjugate()
```

Pour en avoir une version « calcul sans pertes » (lorsque `A` est à entrées dans $\mathbb{Q}$ ou dans $\mathbb{Q} \oplus i\mathbb{Q}$, voire $\bar{\mathbb{Q}}$), il suffit d'éliminer la première boucle `if...else...` du code et de travailler avec `corps=QQbar`.

2.3.5. Application à la diagonalisation des matrices symétriques réelles ou complexes hermitiennes dans une base orthonormée (algorithme de J. Francis & V. Kublanovskaya). On rappelle ici (voir le cours d'Algèbre 2 en L2) que toute matrice symétrique réelle est diagonalisable dans une base orthonormée de $\mathbb{R}^{\mathtt{N}}$, c'est-à-dire s'exprime sous la forme `A = Q*D*Q`*, où `Q` est orthogonale (^{t}`Q` = `Q`* = `Q`$^{-1}$) et `D` est diagonale réelle. La diagonale de `D` consiste en les valeurs propres de `A`, tandis que les vecteurs colonnes de la matrice orthogonale réelle `Q` sont les vecteurs propres associés.

Ce résultat majeur subsiste dans le cadre de $\mathbb{C}^{\mathtt{N}}$ pour les matrices hermitiennes complexes (`A` = `A`* = `(A.transpose()).conjugate()`). La matrice `Q` est dans ce cas unitaire, c'est-à-dire telle que `Q`* = `Q`$^{-1}$. La matrice diagonale `D` est encore dans ce cadre diagonale réelle.

Nous allons présenter dans cette section l'algorithme itératif de John Francis et Vera Kublanovskaya (1961) retournant `Q` et `D` de manière approchée. De fait, cet algorithme pourrait être modifié pour retourner (au moins dans le cadre où les entrées de la matrice en question sont génériques et cette matrice diagonalisable comme matrice à entrées complexes, avec de plus les $\mathrm{Re}(\lambda_j)$, $j = 1, ..., \mathtt{N}$, toutes de valeur absolue distinctes deux à deux) le spectre complexe approché de toute matrice carrée complexe `A`. Nous nous contenterons dans ce cours du cas des matrices symétriques réelles ou hermitiennes complexes qui dans la pratique constitue un cas particulier très fréquent au niveau des applications.

La démarche est la suivante :

- On calcule dans un premier temps la factorisation de Hessenberg `A=Q*H*Q`* de `A` (voir la sous-section précédente 2.3.4) ; ce calcul peut être conduit de manière exacte lorsque les entrées de `A` sont dans le corps des nombres algébriques (par exemple dans $\mathbb{Q}$ ou dans $\mathbb{Q} \oplus i\mathbb{Q}$).
- On lance ensuite sur `H` l'algorithme `Q*H*Q`* de John Francis et Vera Kublanovskaya (voir aussi la sous-section précédente 2.3.4), opération que l'on itère, en la couplant avec un principe de *déflation* que l'on va expliquer plus loin. Au

fil des itérations, la matrice (initialement trigonale) devient de plus en plus « diagonale » et la factorisation « approchée » `A = Q*D*Q`* voulue en résulte.

Voici ici le code `Sage` dans le cadre numérique (on se contentera de ce cadre ici puisque de toutes façons il ne s'agit que d'un algorithme itératif conduisant inéluctablement à des résultats approchés).

```
def DIAGHERMITIENNE1(corps,A,N,iter):
    H = HESSENBERG(corps,A,N)
    Anew = H[0]; Qnew = H[1]
    VP   = []
    for j in range(N-1):
        Qnewaux = identity_matrix(N-j)
        for k in range(iter):
            mu = Anew[N-1-j,N-1-j]
#deux lignes du code impliquant la procedure shift-deflation
            B  = DecompQR(corps, Anew - mu*identity_matrix(N-j),N-j)
            Anew = B[1]*B[0] + mu*identity_matrix(N-j)
#%%%%%%%%%%%%%%%%%
            Qnewaux = Qnewaux*B[0]
        VP = [real(Anew[N-1-j,N-1-j])] + VP
        Anew = Anew.matrix_from_rows_and_columns
               (range(N-j-1),range(N-j-1))
        LQnewaux1 = [Qnewaux[l].list()
        + [0 for ll in range(j)] for l in range(N-j)]
        LQnewaux2 = [[0 for ll in range(N-j+l)]
        + [1] + [0 for ll in range(j-l-1)] for l in range(j)]
        Qnew = Qnew*Matrix(corps, LQnewaux1 + LQnewaux2)
    return diagonal_matrix([real(Anew[0,0])] + VP), Qnew
```

En modifiant la première ligne du code, on peut utilser un calcul exact dans `QQbar` pour calculer `H[0]`, avant de convertir ensuite la première sortie obtenue `H[0]` à l'environnement `ComplexField(Ndigits)`. Il convient cependant pour cela que les entrées de `A` soient algébriques (par exemple dans $\mathbb{Q}$ ou $\mathbb{Q} \oplus i\mathbb{Q}$).

Avant d'expliquer cet algorithme, nous allons énoncer deux principes simples concernant les spectres complexes, celui dit « du *shift* » ou encore du « décalage » et celui dit de « déflation ».

LEMME 2.1 (principe du « *shift* »). *Soit λ est une valeur propre d'une matrice carrée* `A` *de taille* `[N,N]` *(à entrées dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) et $\mathtt{v} \in \mathbb{K}^{\mathtt{N}} \setminus \{0\}$ un vecteur non nul du sous-espace propre E_λ. Alors, pour tout nombre complexe λ_0, $\lambda - \lambda_0$ est valeur propre de $\mathtt{A} - \lambda_0\,\mathtt{matrix.identity(N)}$ avec le même vecteur propre* `v`.

DÉMONSTRATION. On a en effet

$$(\mathtt{A} - \lambda_0 * \mathtt{matrix.identity(N)}) * \mathtt{v} = \mathtt{A} * \mathtt{v} - \lambda_0 * \mathtt{v} = (\lambda - \lambda_0) * \mathtt{v},$$

d'où le résultat. □

C'est le principe dit de *déflation* qui permet en algèbre linéaire, pour une matrice carrée `A` à entrées complexes dont une seule valeur propre se trouve sur le cercle

de rayon $\rho(\mathtt{A})$, de calculer, une fois cette valeur propre et un vecteur propre associé trouvés (par exemple asymptotiquement par une méthode itérative), asymptotiquement également la valeur propre de plus grand module $< \rho(\mathtt{A})$ et un vecteur propre associé (lorsque cette valeur propre se trouve être unique). Voici ce principe simple :

LEMME 2.2 (principe de « *déflation* »). *Soit*

$$\mathtt{A} = \begin{bmatrix} \mathtt{A1(m,m)} & \mathtt{A2(m,N-m)} \\ \mathtt{zeros(N-m,m)} & \mathtt{A3(N-m,N-m)} \end{bmatrix}$$

une matrice « blocs » de taille `[N,N]` *avec* `m<N`. *Alors :*

— *si* λ *est valeur propre de* `A1` *avec un vecteur propre associé* $\mathtt{v}_1$ *dans* $\mathbb{K}^{\mathtt{m}} \setminus \{0\}$, λ *est aussi valeur propre de* `A` *avec comme vecteur propre le vecteur*

$$\mathtt{v} = \mathtt{vector(vlist()+\ [0\ for\ k\ in\ range(N-m)])}\ ;$$

— *si* μ *est une valeur propre de* `A3`, μ *est aussi valeur propre de* `A`.

DÉMONSTRATION. Le premier point est immédiat car

$$\mathtt{A} * \mathtt{v} = \mathtt{A1} * \mathtt{v}_1 = \lambda * \mathtt{v}_1 = \lambda * \mathtt{v}.$$

Le second point est aussi facile : soit $\mathtt{w} \in \mathbb{K}^{\mathtt{N-m}} \setminus \{0\}$ un vecteur propre de `A3` pour cette valeur propre μ. Si μ n'était pas aussi valeur propre de `A1`, la matrice

$$\mathtt{A1} - \mu * \mathtt{matrix.identity(m)}$$

serait inversible et il existerait donc un vecteur `v` de $\mathbb{K}^{\mathtt{m}}$ avec $\mathtt{A3*w} + \mathtt{A1*v} = \mu * \mathtt{v}$. Alors μ serait valeur propre de `A`, un vecteur propre associé étant alors le vecteur `vector(v.list() + w.list())` (non nul dans $\mathbb{K}^{\mathtt{N}}$). Le cas où μ est aussi valeur propre de `A1` nous ramène au cas du premier point. □

2.4. Décomposition en valeurs singulières (`svd`) ; notion de pseudo-inverse

Dans cette section, le corps $\mathbb{K}$ sera $\mathbb{R}$ (respectivement $\mathbb{C}$), considéré ici soit comme l'environnement numérique `RDF` (respectivement `CDF`), correspondant aux flottants réels (respectivement complexes) en double précision, soit comme l'environnement numérique `RealField(Ndigits)` (respectivement `ComplexField(Ndigits)`) si l'on souhaite une précision arbitraire de `Ndigits` bits (en binaire) après la virgule lorsque les nombres sont approchés par des nombres exprimés en virgule flottante.

2.4.1. Décomposition en valeurs singulières d'une matrice rectangulaire à entrées réelles ou complexes. Le résultat central de cette section est le suivant.

THEORÈME 2.3 (décomposition en valeurs singulières d'une matrice de taille `[M,N]` à entrées réelles ou complexes). *Soit* `A` *une matrice de taille* `[M,N]` *à entrées dans* $\mathbb{K}$ *(*$\mathbb{R}$ *ou* $\mathbb{C}$*), de rang* $\mathtt{r} \leq \min(\mathtt{M}, \mathtt{N})$. *Il existe*

— *une matrice* `P` *de taille* `[M,M]` *telle que* `(P.transpose()).conjugate()*P` = `matrix.identity(M)` [1]

1. Ceci signifie que les vecteurs-colonnes de `P` constituent une base orthonormée de l'espace vectoriel $\mathbb{K}^{\mathtt{M}}$ pour le produit scalaire canonique sur ce $\mathbb{K}$-espace vectoriel.

— *une matrice* `Q` *de taille* `[N,N]` *telle que* `(Q.transpose()).conjugate()*Q = matrix.identity(N)` [1]

— *une matrice diagonale réelle*

$$\texttt{D = diagonal\_matrix}([\sigma_0, ..., \sigma_{\texttt{r}-1}])$$

de taille `[r,r]` *à entrées des nombres strictement positifs*

$$\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{r-1} > 0$$

(ne dépendant que de la matrice `A` *et appelés valeurs singulières de cette matrice)*

de manière à ce que

$$\texttt{A} = \texttt{P * DD * (Q.transpose().conjugate())}\,, \tag{2.15}$$

où [2]

$$\texttt{DD = [D zeros(r,N-r); zeros(M-r,N)]}$$

désigne la matrice de taille `[M,N]` *obtenue en complétant* `D` *par des zéros si nécessaire à droite et inférieurement.*

DÉMONSTRATION. Nous allons rédiger la preuve de ce résultat tout en élaborant (sous `Sage`) le code correspondant à son implémentation. Il convient de se souvenir que la notation T' (pour une matrice à entrées complexes), empruntée aux logiciels de calcul scientifique que sont `Scilab` ou `MATLAB`, désigne la *trans-conjuguée* (ou adjointe T^*) de T, soit `(T.transpose()).conjugate()` sous `Sage`, et non simplement sa transposée `T.transpose()` (comme c'est le cas pour une matrice à entrées réelles). Nous nous placerons ici uniquement dans le cas où `N` est supérieur ou égal à `M` ; si ce n'est pas le cas, on remplace `A` par sa transposée avant de transposer au final le résultat obtenu.

On suppose dans un premier temps que le rang de la matrice est égal à `M`, ce qui revient à supposer que l'application linéaire que cette matrice représente dans les bases canoniques de $\mathbb{K}^{\texttt{N}}$ à la source et de $\mathbb{K}^{\texttt{M}}$ au but est surjective. Considérons les `M` vecteurs-lignes de la matrice `A` (c'est-à-dire les images des vecteurs de la base canonique de $\mathbb{K}^{\texttt{N}}$ par l'application linéaire représentée dans les bases canoniques à la source et au but par la matrice `A'`). Ils forment un système libre dans $\mathbb{K}^{\texttt{N}}$ puisque l'on a $\mathrm{rang}(\texttt{A}) = \mathrm{rang}(\texttt{A}') = \min(\texttt{M}, \texttt{N}) = \texttt{M}$. On peut compléter ce système en une base de $\mathbb{K}^{\texttt{N}}$ en choisissant par exemple $\texttt{N} - \texttt{M}$ vecteurs générés aléatoirement dans $\mathbb{R}^{\texttt{N}}$, puis orthonormaliser ensuite le système obtenu suivant le procédé d'orthonormalisation de Gram-Schmidt. La matrice de passage `P0` (dont les vecteurs-colonnes figurent les vecteurs de cette base orthonormée de $\mathbb{K}^{\texttt{N}}$, exprimés dans la base canonique) s'obtient *via* la routine auxiliaire suivante, où l'on fait ici appel sous `Sage` au code `GRAMSCHMIDT` élaboré à la section 2.3.1 (la tolérance `tol` préside ici à la génération aléatoire des vecteurs complétant le système des `M` vecteurs-lignes de `A` que l'on prétend transformer en les `M` premiers vecteurs d'une base orthonormée de $\mathbb{K}^{\texttt{N}}$) :

1. Ceci signifie que les vecteurs-colonnes de `Q` constituent une base orthonormée de l'espace vectoriel $\mathbb{K}^{\texttt{N}}$ pour le produit scalaire canonique sur ce $\mathbb{K}$-espace vectoriel.

2. Il est commode d'utiliser ici la syntaxe de `Scilab` ou de `MATLAB`.

```
sage:
def SVDAUX(A,M,N,corps,tol):
    R  = corps
    if corps == CDF:
        corpsR = RDF
    else:
        corpsR = corps.base_ring()
    L  = []
    for j in range(M):
        L = L + [A[j]]
    for j in range(N-M):
        l = [corpsR.random_element(-tol,tol)
            for k in range(N)]
        L = L + [vector(l)]
    Lo = GRAMSCHMIDT(corps,L)
    return Matrix(Lo[0]).transpose()
```

La matrice `P0` retournée par ce code est une matrice orthogonale (ses vecteurs-colonnes formant une base orthonormée) et la matrice `A*P0` figure la matrice de l'application linéaire représentée par `A` lorsque la base de l'espace but $\mathbb{K}^{\mathtt{M}}$ reste la base canonique, mais que la base de l'espace source $\mathbb{K}^{\mathtt{N}}$ est cette fois choisie comme la base dont les vecteurs sont les entrées de la liste `Lo` constituant le système orthonormé ainsi construit. Du fait que le noyau de l'opérateur linéaire représenté par `A` coïncide avec l'orthogonal de l'image de son adjoint (représenté, lui, par $\mathtt{A}'$) et que l'on a obtenu précisément la liste `Lo` en orthonormalisant une liste dont les `M` premiers éléments étaient les `M` vecteurs-colonnes de la matrice `A'=A.transpose().conjugate()`, on constate que les `N-M` dernières colonnes de la matrice `A*P0` sont nulles, tandis que les `M` premières colonnes forment une matrice inversible de taille `[M,M]` que l'on notera `A0`. Cette matrice `A0` est d'ailleurs triangulaire supérieure puisque les `M` premiers vecteurs du système que l'on a orthonormalisé (suivant le procédé « triangulaire » de Gram-Schmidt) étaient les `M` vecteurs-colonnes de la matrice `A'`. Quitte à multiplier à droite `A` par la matrice orthogonale `P0`, on peut supposer que les `N-M` dernières colonnes de la matrice `A` sont nulles, ce que l'on supposera à compter de maintenant en désignant toutefois par `A` à partir de maintenant la matrice $\mathtt{Anew} = \mathtt{A} * \mathtt{P0}$.

Si nous supposons le résultat acquis (c'est-à-dire la décomposition (2.15) validée), on observe que

$$\mathtt{A*A'} = \mathtt{Anew*Anew'} = \mathtt{P*DD*Q'*Q*DD'*P'} = \mathtt{P*DD*DD*P'}. \tag{2.16}$$

Ainsi donc la relation (2.16) se présente comme la réduction normale de la matrice symétrique réelle ou hermitienne complexe `A*A'= Anew*Anew'`. On note que les valeurs propres (réelles) de cette matrice hermitienne sont bien positives ou nulles puisque, pour tout vecteur `V` de $\mathbb{K}^{\mathtt{N}}$, on a

$$\mathtt{V.conjugate()*(Anew*Anew')*V} = \|\mathtt{Anew'*V}\|^2 \geq 0.$$

La matrice $\mathtt{A} * \mathtt{A}' = \mathtt{Anew*Anew'}$ est en fait, compte tenu de la forme de la matrice `Anew`, la matrice bloc

```
Anew*Anew' = [A0*A0' zeros(M,N-M); zeros(N-M,N)]
```

si l'on utilise les notations de Scilab ou MATLAB. Dans le cas où le rang de A est maximal et égal donc ici à M, l'instruction [1]

```
sage:
AA0  = A0*(A0.transpose().conjugate())
PP0  = DIAGHERMITIENNE1(corps,AA0,M,iter)[1]
```

retourne une matrice orthogonale ou unitaire PP0 de taille [M,M] telle que

PP0'*(A0*A0')*PP0

soit une matrice diagonale dont les entrées sont des nombres réels positifs ou nuls, à savoir précisément ici les M valeurs propres $\sigma_0^2 \geq \cdots \geq \sigma_{M-1}^2 > 0$ de la matrice symétrique réelle inversible A0*A0'. On observe dans ce cas que

A0*A0' = (PP0*D0)*(PP0*D0)'

où D0 désigne la matrice diagonale dont les entrées sont les racines strictement positives des nombres $\sigma_0^2 \geq \cdots \geq \sigma_{M-1}^2 > 0$. Cette matrice D0 s'obtient ainsi à partir de PP0 et AA0=A0*A0' :

```
sage:
DD0   = DIAGHERMITIENNE1(corps,AA0,M,iter)[0]
L     = DD0.diagonal();
sqrtL = [sqrt(real(L[k])) for k in range(M)]
D0    = diagonal_matrix(sqrtL)
```

Il reste à remarquer que, du fait que

AA0 = A0*A0' = (PP0*D0)*(PP0*D0)',

la matrice A0'*([(PP0*D0)']$^{-1}$ = A0'*PP0*D0^{-1} est une matrice orthogonale Q0 de taille [M,M]. Si l'on note

QQ = [Q0 zeros(M,N-M); zeros(M,N-M) eyes(N-M)],

on voit que QQ est aussi orthogonale réelle ou unitaire complexe et que l'on a la relation

Anew = PP0 * [D0 zeros(N-M,N-M)] * QQ'.

Revenant à A, on trouve ainsi

A = PP0 * [D0 zeros(N-M,N-M)] * QQ'*P0',

ce qui est dans ce cas la forme voulue.

Lorsque le rang de la matrice A est strictement inférieur à M = min(M, N), on raisonne de manière identique, à ceci près que le processus d'orthogonalisation du système constitué des M vecteurs-colonnes de la matrice A' ne peut être mené à terme ; on ne peut que construire un système orthonormé de r vecteurs (constituant une base de l'image de A' dans $\mathbb{K}^N$) qu'il convient ensuite de compléter en un système orthonormé de $\mathbb{K}^N$. La matrice Anew = A*P0 se présente cette fois sous la forme

Anew = [A0 zeros(r,N); zeros(M-r,N)]

où A0 est une matrice inversible de taille [r,r]. On poursuit ensuite les calculs en réduisant normalement A0*A0.transpose().conjugate() comme dans le cas précédemment envisagé. □

1. Pour ce qui est de la routine DIAGHERMITIENNE1, on se reportera à la section 2.3.5 ci-dessus.

Le code récapitulant ces diverses opérations et retournant (dans cet ordre) les matrices `P`, `DD` et `Q` est décrit ici. Les valeurs singulières ne sont toutefois pas ici retournées toujours dans l'ordre décroissant du fait de l'algorithme utilisé pour la réduction des matrices symétriques réelles ou hermitiennes complexes (algorithme `QR` de Francis - Kublanovskaya). Cette routine n'est en principe vraiment opérationnelle que si le rang de la matrice `A` est maximal, c'est-à-dire égal à $\min(\mathtt{M},\mathtt{N})$ (soit à `M` lorsque $\mathtt{N} \geq \mathtt{M}$) ; pouvoir décider en effet si oui ou non une des valeurs singulières σ_j est nulle demeure hors de portée dans le cadre du calcul avec pertes (ici avec seulement une précision de `Ndigits`). Voici le code en question.

```
sage:
def VALEURSSINGULIERES(A,M,N,corps,tol,iter):
    if N < M:
        AA = (A.transpose()).conjugate()
        MM = N; NN = M
    else:
        AA = A; MM = M; NN = N
    P0     = SVDAUX(AA,MM,NN,corps,tol)
    Anew   = AA*P0
    LA0    = [Anew.transpose()[k] for k in range(MM)]
    A0     = Matrix(LA0).transpose()
    AA0    = A0*(A0.transpose().conjugate())
    DIAG   = DIAGHERMITIENNE1(corps,AA0,MM,iter)
    DD0    = DIAG[0]; PP0 = DIAG[1]
    L      = DD0.diagonal();
    sqrtL  = [sqrt(real(L[k])) for k in range(MM)]
    D0     = diagonal_matrix(sqrtL)
    LDD0   = []
    for j in range(MM):
        LDD0 = LDD0 + [vector(D0[j].list()
               + [0 for k in range(NN-MM)])]
    DD0    = Matrix(LDD0)
    Q0     = ((A0.transpose()).conjugate())*PP0*D0^(-1)
    LQQ    = []
    for j in range(MM):
        LQQ = LQQ + [vector(Q0[j].list()
              + [0 for k in range(NN-MM)])]
    for j in range(NN-MM):
        LQQ = LQQ + [identity_matrix(NN)[MM+j]]
    QQ     = Matrix(LQQ)
    if N >= M:
        return PP0, DD0, P0*QQ
    else:
        return P0*QQ, (DD0.conjugate()).transpose(), PP0
```

Le lancement sous `Scilab` ou `MATLAB` du programme retournant les trois sorties `P`, `DD`, `Q` impliquées dans une décomposition (2.15) se fait suivant l'instruction :

```
>>  [P,DD,Q] = svd(A);
--> [P,DD,Q] = svd(A);
```

Mais les calculs ici ne sont effectués qu'en double précision (ce qui correspond à `corps = RDF` ou `CDF` sous `Sage`), ce qui est très loin d'être satisfaisant dans des problèmes où une très haute précision s'avère requise.

2.4.2. Pseudo-inverse d'une application $\mathbb{R}$ ou $\mathbb{C}$-linéaire. Soient $\mathtt{M}$ et $\mathtt{N}$ deux entiers strictement positifs tels que $\mathtt{M} \leq \mathtt{N}$ et L une application $\mathbb{K}$-linéaire de $\mathbb{K}^{\mathtt{N}}$ dans $\mathbb{K}^{\mathtt{M}}$, représentée (lorsque les espaces vectoriels source $\mathbb{K}^{\mathtt{N}}$ et but $\mathbb{K}^{\mathtt{M}}$ sont rapportés à leurs bases canoniques respectives) par une matrice rectangulaire $\mathtt{A}$ à $\mathtt{M}$ lignes et $\mathtt{N}$-colonnes. On suppose ici que l'application linéaire L est surjective, c'est-à-dire de rang $\min(\mathtt{M},\mathtt{N}) = \mathtt{M}$.

Si $\mathtt{Y} \in \mathbb{K}^{\mathtt{M}}$ est un vecteur de l'espace-but, le $\mathbb{K}$-sous-espace affine de $\mathbb{K}^{\mathtt{M}}$ défini comme

$$L^{-1}(\{\mathtt{Y}\}) := \{\mathtt{X} \in \mathbb{K}^{\mathtt{N}}\,;\, L(\mathtt{X}) = \mathtt{Y}\}$$

est un $\mathbb{K}$-sous-espace affine de l'espace source $\mathbb{K}^{\mathtt{M}}$ dont on est certain qu'il est toujours non vide puisque L est supposée surjective. D'après la formule du rang, on a d'ailleurs $\dim_{\mathbb{K}} L^{-1}(\{\mathtt{Y}\}) = \mathtt{N} - \mathtt{M}$.

Définition 2.5 (pseudo-inverse d'un vecteur sous l'action d'une transformation linéaire surjective). La projection orthogonale du vecteur $\mathtt{0} \in \mathbb{K}^{\mathtt{N}}$ sur le $\mathbb{K}$-sous espace affine $L^{-1}(\{\mathtt{Y}\})$ représente le vecteur de norme $\|\ \|_2$ minimale parmi tous les vecteurs $\mathtt{X}$ de l'epace source $\mathbb{K}^{\mathtt{N}}$ tels que $L(\mathtt{X}) = \mathtt{Y}$. On appelle cet élément $\mathrm{Proj}_{L^{-1}(\{\mathtt{Y}\})}[\mathtt{0}]$ le *pseudo-inverse* de $\mathtt{Y}$ sous l'action de la transformation linéaire surjective L. L'application $\mathtt{Y} \in \mathbb{K}^{\mathtt{M}} \longmapsto \mathtt{X} \in \mathbb{K}^{\mathtt{N}}$ est en fait une application $\mathbb{K}$-linéaire que l'on appelle pseudo-inverse de l'application $\mathbb{K}$-linéaire L.

La décomposition en valeurs singulières de la matrice $\mathtt{A}$ permet de calculer aisément le pseudo-inverse d'un vecteur $\mathtt{Y}$ donné dans l'espace but. C'est d'ailleurs aussi en utilisant la décomposition en valeurs singulières de la matrice $\mathtt{A}$ que l'on voit que l'application qui à un vecteur $\mathtt{Y}$ de l'espace source $\mathbb{K}^{\mathtt{N}}$ associe son pseudo-inverse $\mathtt{X}$ est bien une application $\mathbb{K}$-linéaire dont on va même en fait calculer la matrice. On rappelle que l'algorithme de décomposition en valeurs singulières d'une matrice rectangulaire $\mathtt{A}$, de taille `[M,N]` avec $\mathtt{M} \leq \mathtt{N}$ et de rang $(\mathtt{M})$ maximal, retourne deux matrices $\mathtt{P}$ et $\mathtt{Q}$ toutes deux orthogonales réelles (si $\mathbb{K} = \mathbb{R}$) ou unitaires (si $\mathbb{K} = \mathbb{C}$) respectivement de taille $[\mathtt{M},\mathtt{M}]$ et $[\mathtt{N},\mathtt{N}]$, ainsi que la liste

$$\sigma_0 \geq \sigma_1 \geq \cdots \geq \sigma_{\mathtt{M}-1} > 0$$

des valeurs singulières de $\mathtt{A}$, constituant la liste des éléments diagonaux de la sortie $\mathtt{DD}$ (seconde sortie retournée par le code). Voici donc comment on procède pour calculer le pseudo-inverse d'un vecteur $\mathtt{Y}$ donné dans l'espace-but $\mathbb{K}^{\mathtt{M}}$ (et prouver au passage qu'il s'agit bien d'une opération $\mathbb{K}$-linéaire).

— On exprime le vecteur $\mathtt{Y}$ dans la base (orthonormée) de $\mathbb{K}^{\mathtt{M}}$ constituée des $\mathtt{M}$ vecteurs colonne de la matrice $\mathtt{P}$, notés $\mathtt{v}_0, ..., \mathtt{v}_{\mathtt{M}-1}$, ce qui donne :

$$\mathtt{Y} = \sum_{j=0}^{\mathtt{M}-1} \langle \mathtt{Y}, \mathtt{v}_j \rangle\, \mathtt{v}_j.$$

— Si les valeurs singulières $\sigma_0, ..., \sigma_{\mathtt{M}-1}$ sont retournées dans cet ordre, on réalise le pseudo-inverse de `Y` comme le vecteur

$$\mathtt{X} := \sum_{j=0}^{\mathtt{M}-1} \frac{\langle \mathtt{Y}, \mathtt{v}_j \rangle}{\sigma_j} \mathtt{w}_j,$$

où cette fois $\mathtt{w}_0, ..., \mathtt{w}_{\mathtt{M}-1}$ sont les `M` premiers vecteurs colonne de la matrice `Q`. Ceci revient à dire que l'application linéaire L^{-1}_{pseudo} de $\mathbb{K}^{\mathtt{M}}$ dans $\mathbb{K}^{\mathtt{N}}$ est représentée (lorsque l'espace source cette fois $\mathbb{K}^{\mathtt{M}}$ et l'espace but cette fois $\mathbb{K}^{\mathtt{N}}$ sont rapportés à leurs bases canoniques respectives) par la matrice :

$$\mathtt{Q}\,\big(\texttt{([Dinv zeros(M,N-M)]).transpose()}\big)\,\mathtt{P}^*$$

où

$$\mathtt{Dinv} = \texttt{diagonal\_matrix}([\sigma_0^{-1}, ..., \sigma_{\mathtt{M}-1}^{-1}]).$$

On vérifie immédiatement que `A(Ainv(X)) = Y`, ce qui prouve que le vecteur ansi construit `X = Yinv` est bien dans $L^{-1}(\{\mathtt{Y}\})$. Du fait que les applications linéaires représentées par les matrices `P` et `Q` correspondent à des applications respectant la norme euclidienne respectivement dans $\mathbb{K}^{\mathtt{M}}$ et dans $\mathbb{K}^{\mathtt{N}}$, on peut se ramener, pour vérifier que le vecteur `X` ainsi construit est bien de norme euclidienne minimale parmi tous les vecteurs de $\mathbb{K}^{\mathtt{N}}$ appartenant à $L^{-1}(\{\mathtt{Y}\})$, au cas où la matrice `A` est la matrice `DD` de taille `[M,N]` (dont les seules entrées non nulles sont les entrées situées sur la diagonale, valant précisément $\sigma_0, ..., \sigma_{\mathtt{M}-1}$) retournée comme seconde sortie par l'algorithme de décomposition en valeurs singulières. Si l'application L se trouve représentée par une telle matrice

```
DD = [D zeros(M,N-M)]
```

il est immédiat de constater que

```
Yinv = ([Dinv zeros(M,N-M)].transpose()) Y
```

est bien le vecteur de norme euclidienne minimale parmi tous les vecteurs `X` tels que

```
DD X = Y.
```

Remarque 2.2 (le cas où `M`, `N` et le rang de la matrice `A` sont quelconques)**.** Dans le cas où l'on ne fait aucune hypothèse sur `M`, `N` et le rang $r \leq \min(\mathtt{M}, \mathtt{N})$ de la matrice `A`, on appelle, étant donné un vecteur quelconque `Y` de l'espace but $\mathbb{K}^{\mathtt{M}}$, *pseudo-inverse* de `Y` sous l'action de l'opérateur $\mathbb{K}$-linéaire L représenté par la matrice `A`, le vecteur

$$\mathtt{X} := \sum_{\{0 \leq j \leq \inf(\mathtt{M},\mathtt{N})-1\,;\,\sigma_j > 0\}} \frac{\langle \mathtt{Y}, \mathtt{v}_j \rangle}{\sigma_j} \mathtt{w}_j,$$

où les $\mathtt{v}_j$, $j = 0, ..., \mathtt{M}-1$, sont les vecteurs colonnes de la matrice `P`, $\mathtt{w}_j$, $j = 0, ..., \mathtt{N}-1$ les vecteurs colonnes de la matrice `Q`, `P` et `Q` désignant les première et troisième sorties retournées par l'algorithme de décomposition en valeurs singulières (`A = P*DD*Q`*). Ce vecteur `X` est encore noté $L^{-1}_{\text{pseudo}}(\mathtt{Y})$ et l'application linéaire qui a $\mathtt{Y} \in \mathbb{K}^{\mathtt{M}}$ associe $L^{-1}_{\text{pseudo}}(\mathtt{Y}) \in \mathbb{K}^{\mathtt{N}}$ est représentée encore par la matrice

$$\begin{cases} \mathtt{Q}\,\big(\texttt{([Dinv zeros(M,N-M)]).transpose()}\big)\,\mathtt{P}^* & \text{si } \mathtt{M} \leq \mathtt{N} \\ \mathtt{Q}\,\big(\texttt{([Dinv; zeros(M-N,N)]).transpose()}\big)\,\mathtt{P}^* & \text{si } \mathtt{N} < \mathtt{M}, \end{cases}$$

où `Dinv` est cette fois définie par

$$\texttt{diagonal\_matrix}([\tau_0, ..., \tau_{\inf(\texttt{M},\texttt{N})-1}])$$

avec

$$\tau_j = \begin{cases} \sigma_j^{-1} \text{ lorsque } \sigma_j > 0 \\ 0 \quad \text{si } \sigma_j = 0. \end{cases}$$

Le pseudo-inverse $L^{-1}_{\text{pseudo}}(\texttt{Y})$ de $\texttt{Y}$ ne vérifie cette fois plus $L(L^{-1}_{\text{pseudo}}(\texttt{Y})) = \texttt{Y}$ (il n'y a plus de raison pour que L soit ici surjective), mais c'est le point $\texttt{X}_{\min}$ de $\mathbb{K}^{\texttt{N}}$ de norme euclidienne minimale parmi tous les points de l'espace-source où la fonction

$$\texttt{X} \in \mathbb{K}^{\texttt{N}} \longmapsto \|\texttt{Y} - L(\texttt{X})\|_2$$

réalise son minimum (ce minimum étant la distance euclidienne de $\texttt{Y}$ au sous-espace vectoriel $L(\mathbb{K}^{\texttt{N}})$). Cette notion joue un rôle très important dans les problèmes pratiques. Il est en effet souvent très important de connaitre, étant donné un vecteur $\texttt{Y}$ de $\mathbb{K}^{\texttt{M}}$ ($\texttt{M} \geq 1$) dont on sait qu'il se trouve à une distance $d \geq 0$ du sous-espace image $L(\mathbb{K}^{\texttt{N}})$ par une application linéaire $L : \mathbb{K}^{\texttt{N}} \to \mathbb{K}^{\texttt{M}}$ ($\texttt{N} \geq 1$) donnée, LE vecteur $\texttt{X}$ de plus petite norme euclidienne parmi tous les vecteurs de $\mathbb{K}^{\texttt{N}}$ tels que $\|L(\texttt{X}) - \texttt{Y}\|_2 = d$. Ce vecteur est précisément $L^{-1}_{\text{pseudo}}(\texttt{Y})$. Lorsque $d = 0$, ce vecteur est bien sûr dans $L^{-1}(\{Y\})$, et c'est justement LE vecteur de $\mathbb{K}^{\texttt{N}}$ de norme euclidienne minimale parmi tous les vecteurs $\texttt{X}$ solutions du système linéaire $L(\texttt{X}) = \texttt{Y}$.

2.4.3. Calcul du conditionnement d'une matrice rectangulaire par rapport à la norme euclidienne.

DÉFINITION 2.6 (conditionnement d'une matrice rectangulaire par rapport à la nore euclidienne). Le conditionnement d'une matrice rectangulaire $\texttt{A}$ à entrées réelles ou complexes par rapport à la norme euclidienne est défini comme le quotient $\sigma_0/\sigma_{\texttt{r}-1}$ de la plus grande des valeurs singulières par la plus petite des valeurs singulières non nulles.

On remarque que le conditionnement d'une matrice rectangulaire $\texttt{A}$ à $\texttt{M}$ lignes et $\texttt{N}$ colonnes s'exprime comme le produit

$$\text{cond}_{\|\ \|_2}(\texttt{A}) = \|\texttt{A}\|_2 \times \|\texttt{A}^{-1}_{\text{pseudo}}\|_2 \,,$$

où

$$\|\texttt{A}\|_2 = \sup_{\texttt{v} \in \mathbb{K}^{\texttt{N}} \setminus \{0\}} \frac{\|\texttt{A}*\texttt{v}\|_2}{\|\texttt{v}\|_2} \quad \text{et} \quad \|\texttt{A}^{-1}_{\text{pseudo}}\|_2 = \sup_{\texttt{v} \in \mathbb{K}^{\texttt{M}} \setminus \{0\}} \frac{\|\texttt{A}^{-1}_{\text{pseudo}} * \texttt{v}\|_2}{\|\texttt{v}\|_2},$$

où $\texttt{A}^{-1}_{\text{pseudo}}$ est la matrice à $\texttt{N}$ lignes et $\texttt{M}$ colonnes représentant le pseudo-inverse $L^{-1}_{\texttt{A},\text{pseudo}}$ de l'application linéaire $L_{\texttt{A}}$ représentée par $\texttt{A}$ lorsque $\mathbb{K}^{\texttt{M}}$ et $\mathbb{K}^{\texttt{N}}$ sont rapportés à leurs bases canoniques respectives.

CHAPITRE 3

Suites de matrices et méthodes itératives

Si $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$, le $\mathbb{K}$-espace vectoriel $\mathscr{M}(\mathbb{K}; \mathtt{N}, \mathtt{N})$ des matrices carrées de taille `[N,N]` à entrées dans $\mathbb{K}$ est un $\mathbb{K}$-espace-vectoriel de dimension $\mathtt{N}^2$, l'organisation en tableaux plutôt qu'en simple ligne (« array » par exemple dans la syntaxe `Python`) n'intervenant que dans un second temps. Ce $\mathbb{K}$-espace vectoriel est donc isomorphe à $\mathbb{K}^{\mathtt{N}^2}$.

Toutes les normes sur $\mathbb{K}^{\mathtt{N}^2}$ sont équivalentes, ce qui signifie qu'étant données deux telles normes $|\ |$ et $\|\ \|$, il existe une constante strictement positive K telle que

$$\forall \mathtt{v} \in \mathbb{K}^{\mathtt{N}}, \qquad \frac{|\mathtt{v}|}{K} \leq \|\mathtt{v}\| \leq K\,|\mathtt{v}|.$$

Si l'on envisage l'incarnation de $\mathbb{K}^{\mathtt{N}^2}$ en $\mathscr{M}(\mathbb{K}; \mathtt{N}, \mathtt{N})$, on distingue les normes matricielles (provenant d'une norme sur $\mathbb{K}^{\mathtt{N}}$, comme les normes de Minkowski $\|\ \|_1$, $\|\ \|_2$, $\|\ \|_\infty$) de celles qui le ne sont pas (par exemple la norme $\|\mathtt{A}\|_{\sup} := \sup |a_{j,k}|_{1 \leq j,k_l eq \mathtt{N}}$. Lorsque nous parlerons dans ce chapitre de « norme » sur le $\mathbb{K}$-espace vectoriel $\mathscr{M}(\mathbb{K}; \mathtt{N}, \mathtt{N})$, nous entendrons, sauf indication contraire, « norme matricielle ».

À toute norme $|\ |$ dans $\mathbb{K}^{\mathtt{N}^2}$, on attache une distance $d = d_{|\ |}$ définie par

$$d_{|\ |}(\mathtt{v}, \mathtt{ww}) := |\mathtt{v} - \mathtt{w}| \quad \forall\, v, w \in \mathbb{K}^{\mathtt{N}^2}.$$

3.1. Suites de Cauchy et théorème du point fixe

Si $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$, le $\mathbb{K}$-espace vectoriel $\mathbb{K}^{\mathtt{N}^2}$ (quelque soit la valeur de `N`, ce peut fort bien être une trentaine de milliards, le nombre estimé de pages web, comme c'est le cas dans le contexte de l'algorithmique régissant les moteurs de recherche sur la toile !), équipé d'une norme arbitraire (peu importe laquelle, elles sont toutes, on l'a rappelé, équivalentes) est un $\mathbb{K}$-*espace vectoriel complet*, c'est-à-dire un $\mathbb{K}$-espace vectoriel dans lequel toute *suite de Cauchy* converge. On précise donc dans ce paragraphe cette notion.

Un espace métrique (c'est-à-dire un ensemble E équipé d'une distance d) est dit *complet* relativement à cette distance d si et seulement si toute suite $(u_k)_{k \geq 0}$ d'éléments de E vérifiant le *critère de Cauchy* relativement à la distance d, c'est-à-dire

$$(3.1)\ \ \forall\, \epsilon > 0,\ \exists K = K(\epsilon) \in \mathbb{N} \text{ tel que } \Big(k_1 \geq K(\epsilon),\ k_2 \geq K(\epsilon)\Big) \Longrightarrow d(u_{k_1}, u_{k_2}) < \epsilon,$$

est convergente dans E (au sens de cette distance) vers un point u_∞ (ce qui signifie $\lim_{k \to +\infty} d(u_k, u_\infty) = 0$). Fait capital pour nous : le $\mathbb{K}$-espace $\mathbb{K}^{\mathtt{N}^2}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$), équipé de la distance $d_{\|\ \|}$ attachée à une norme quelconque $\|\ \|$ se trouve être, puisque $\mathbb{K}$ l'est pour la distance usuelle (attachée à la valeur absolue lorsque $\mathbb{K} = \mathbb{R}$ ou

au module lorsque $\mathbb{K} = \mathbb{C}$), un espace métrique complet. Ceci vaut aussi pour son incarnation $\mathscr{M}(\mathbb{K}; \mathtt{N}, \mathtt{N})$, avec comme distance la distance attachée à n'importe quelle norme matricielle.

Le fil d'Ariane de ce chapitre sera un résultat majeur (dans le cadre de l'espace métrique $E = \mathscr{M}(\mathbb{K}; \mathtt{N}, \mathtt{N})$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$), équipé de la distance d attachée à une norme que l'on supposera (ce qui n'est pas indispensable, mais plus commode) être une norme matricielle : le *théorème du point fixe*. Ce qui fera pour nous la force de ce théorème dans ce cours d'algorithmique algébrique est que sa preuve s'appuie de bout en bout sur une démarche *algorithmique*.

THEORÈME 3.1 (le théorème du point fixe dans un espace métrique complet (E, d)). *Soit (E, d) un espace métrique complet et $T : E \to E$ une application telle qu'il existe une constante $\kappa \in]0, 1[$ réalisant*[1]

$$\forall\, x, y \in E,\ d(T(x), T(y)) \leq \kappa\, d(x, y)\ ; \tag{3.2}$$

une telle application est dite strictement contractante de E dans E. Il existe un unique point x_{fixe} de E invariant (donc restant fixe) sous l'action de T, c'est-à-dire tel que $T(x_{\text{fixe}}) = x_{\text{fixe}}$. De plus, cet unique point fixe x_{fixe} s'obtient comme la limite de n'importe quelle suite $(u_k)_{k\geq 0}$ d'éléments de E, générée à partir d'un point arbitraire x de E ($u_0 = x$) et régie ensuite par la relation inductive

$$u_{k+1} = T(u_k) \quad (k \geq 0). \tag{3.3}$$

On a de plus

$$d(u_k, x_{\text{fixe}}) \leq \frac{\kappa^k}{1-\kappa}\, d(u_1, u_0) = e^{-k \log(1/\kappa)}\, \frac{d(u_1, u_0)}{1-\kappa} \quad \forall\ k \geq 1. \tag{3.4}$$

Autrement dit, la convergence d'une telle suite itérative $(u_k)_{k\geq 0}$ vers l'unique point fixe x_{fixe} de T est une convergence où l'erreur décroit au moins comme une suite géométrique convergente de raison κ, donc exponentiellement en fonction de k lorsque k tend vers l'infini.

DÉMONSTRATION. S'il y avait deux points fixes distincts x_{fixe} et $\tilde{x}_{\text{fixe}}$, on aurait

$$d(T(x_{\text{fixe}}), T(\tilde{x}_{\text{fixe}})) = d(x_{\text{fixe}}, \tilde{x}_{\text{fixe}}) \leq \kappa\, d(x_{\text{fixe}}, \tilde{x}_{\text{fixe}})\ ;$$

ceci est impossible car $\kappa < 1$ et que $d(x_{\text{fixe}}, \tilde{x}_{\text{fixe}}) > 0$ d'après la première contrainte (être « définie positive ») auquel doit se plier la notion de distance. Le point fixe x_{fixe}, si tant est qu'il existe, est donc nécessairement unique.

La suite $(u_k)_{k\geq 0}$ générée à partir de $x \in E$ se plie aux inégalités :

$$d(u_{k+1}, u_k) \leq \kappa\, d(u_k, u_{k-1}) \leq \cdots \leq \kappa^k\, d(u_1, u_0). \tag{3.5}$$

L'inégalité triangulaire (seconde contrainte à laquelle doit se plier la notion de distance) implique alors que, pour tout $k \in \mathbb{N}^*$, pour tout $p \in \mathbb{N}^*$,

$$d(u_{k+p}, u_k) \leq \sum_{\ell=k}^{\ell=k+p-1} d(u_{\ell+1}, u_\ell) \leq \kappa^k \times \sum_{\ell=0}^{p-1} \tau^\ell \times d(u_1, u_0) \leq \frac{\kappa^k}{1-\kappa}\, d(u_1, u_0). \tag{3.6}$$

1. Le choix de $\kappa \geq 1$ ne convient plus. Par exemple, dans l'algorithmique soutendant le fonctionnement de `google`, on s'accorde dans le modèle sur un choix de $\kappa \simeq 0.83$, ce qui s'avère être un juste compromis, voit par exemple [**CalcLog**], section 4.2, pour une présentation du modèle algorithmique régissant les moteurs de recherche sur la toile.

Comme $\lim_{k\to+\infty}\kappa^k = 0$, il en résulte que la suite $(u_k)_{k\geq 0}$ vérifie le critère de Cauchy (3.1) relativement à la distance d et par conséquent converge (au sens de cette distance) vers une limite u_∞ dans E. Le fait que T soit strictement contractante (contractante ici suffirait) implique que l'application T est continue de E dans E (relativement à la topologie d'espace métrique induite par la distance d). Comme $u_{k+1} = T(u_k)$ pour tout $k \geq 0$, on obtient donc en passant à la limite lorsque k tend vers l'infini que l'on a aussi $T(u_\infty) = u_\infty$, et par conséquent que u_∞ est l'unique point fixe de T dans E. En revenant maintenant aux inégalités (3.6), cette fois en « gelant » $k \geq 1$ et en faisant tendre p vers $+\infty$, il résulte de la continuité de la fonction distance que l'on a bien les estimations asymptotiques (3.4). Le résultat est donc acquis. □

3.2. Résolution approchée itérative de systèmes de Cramer

3.2.1. Le contexte général. Supposons que l'on veuille résoudre dans $\mathbb{K}^{\mathtt{N}}$ (ici, on prend toujours $\mathbb{K} = \mathbb{R}$ ou $\mathbb{K} = \mathbb{C}$) un système du type $A * \mathtt{X} = B$, que l'on suppose dans un premier temps ici carré, de taille `[N,N]`, avec la matrice A de plus inversible [1] à entrées dans le corps $\mathbb{K}$ et le second membre B donné dans $\mathbb{K}^{\mathtt{N}}$. Certes on dispose de la formule $\mathtt{X} = A^{-1} * B$. Cependant le calcul direct de A^{-1} (par exemple comme la matrice des cofacteurs de A divisée par le déterminant de cette matrice) induit des calculs en général impossibles car trop coûteux en temps [2] lorsque `N` est très grand, sauf si l'on a de la chance et que la matrice A à le bon goût d'être « creuse », c'est-à-dire de contenir beaucoup d'entrées nulles, comme le sont par exemple les matrices d'adjacence de graphes pondérés intervenant dans la démarche soutendant l'algorithme `Pagerank` (voir [**CalcLog**], section 4.5). Même dans pareils cas cependant, on préfère utiliser l'un des deux types d'attaque suivants :

— une attaque *directe*, telle celle fondant la *méthode du pivot*, dite aussi *méthode de Gauß*, basée sur la résolution directe de systèmes linéaires partiels plus simples (par exemple des systèmes intermédiaires dont la matrice A se présente sous forme triangulaire, inférieure ou supérieure) ; on ne traitera pas ces méthodes ici (on les enseigne souvent en Licence 1) et l'on renvoie le lecteur par exemple à leur présentation détaillée dans [**MathAppL3**] (chapitre 1, section IV.2 [3]).

1. Il s'agit donc d'un système de Cramer.

2. On rappelle que le développement d'un déterminant « plein » de taille `[N,N]` par la règle de Sarrus implique `N!` termes.

3. Contentons nous de signaler que la complexité algorithmique de la méthode du pivot est de l'ordre de $2\mathtt{N}^3/3$ si l'on convient de faire l'hypothèse simplificatrice suivant laquelle toutes les opérations ont même coût (voir par exemple l'exercice 1.12 de [**MathAppL3**]). La méthode de Gauß repose essentiellement sur le fait que toute matrice A de taille `[N ,N ]` dont les mineurs principaux $\det[a_{jk}]_{0\leq j,k\leq m-1}$, $m = 0, ..., \mathtt{N}-1$ sont non nuls se factorise comme le produit $A = \mathtt{L} * \mathtt{U}$ d'une matrice triangulaire inférieure `L` par une matrice triangulaire supérieure `T` ; on a dans ce cas non nullité de tous les pivots utilisés dans la méthode de Gauß. L'apparition de « pivots » nuls au fil de cette procédure se gère à chaque étape par une permutation des équations affines en jeu. Nous reviendrons plus loin dans ce cours sur la description algorithmique des décompositions dites `LU` héritées de la méthode du pivot de Gauß. Il est important de mettre à leur actif que pareilles méthodes directes peuvent être conduites dans le cadre du calcul sans pertes lorsque par exemple la matrice A est déclarée comme matrice à entrées rationnelles (`QQ` dans `Sage`) ou algébriques (`QQbar`

— une approche *itérative* (ou indirecte) consistant précisément (d'où la terminologie) à utiliser une démarche itérative, donc une démarche relevant par nature même du calcul avec pertes et qui se fonde sur l'algorithmique sur laquelle repose la preuve du théorème du point fixe (théorème 3.1).

On choisit ici de s'intéresser au second angle d'attaque.

Le théorème du point fixe implique (théorème 3.1) implique le résultat suivant :

PROPOSITION 3.1 (résolution itérative de $A*\mathtt{X}=B$). *Soit A une matrice de type* `[N,N]` *à entrées dans $\mathbb{K}$ ($\mathbb{K}=\mathbb{R}$ ou $\mathbb{C}$) s'écrivant sous la forme $A=A_1-A_2$, où A_1 et A_2 sont deux matrices de type* `[N,N]` *à coefficients dans $\mathbb{K}$ telles que A_1 soit inversible et que $\rho(A_1^{-1}*A_2)<1$. Alors A et aussi inversible et la suite de vecteurs $(\mathtt{X}_k)_{k\geq 0}$ de $\mathbb{K}^{\mathtt{N}}$ initiée sur un vecteur arbitraire $\mathtt{X}_0$ de $\mathbb{K}^{\mathtt{N}}$ et régie ensuite par l'algorithme itératif*

$$\mathtt{X}_{k+1}=A_1^{-1}*A_2*\mathtt{X}_k+A_1^{-1}*B \qquad \forall\, k\geq 1 \tag{3.7}$$

converge lorsque k tend vers l'infini vers l'unique solution $\mathtt{X}$ du système de Cramer $A*\mathtt{X}=B$. De plus, si $0<\epsilon<1-\rho(A_1^{-1}*A_2)$ et que l'on dispose d'une norme $\|\ \|^{(\epsilon)}$ sur $\mathbb{K}^{\mathtt{N}}$ telle que pour la norme matricielle correspondante on ait

$$\|A_1^{-1}*A_2\|^{(\epsilon)}\leq \rho(A_1^{-1}*A_2)+\epsilon=\kappa_\epsilon<1$$

(ce qui est possible d'après la proposition 1.1), alors on a

$$\|\mathtt{X}_k-\mathtt{X}\|^{(\epsilon)}\leq \kappa_\epsilon^k\,\frac{\|\mathtt{X}_1-\mathtt{X}_0\|^{(\epsilon)}}{1-\kappa_\epsilon} \quad \forall\, k\geq 0, \tag{3.8}$$

et la décroissance exponentielle vers 0 de $\|\mathtt{X}_k-\mathtt{X}\|$ se trouve assurée pour tout choix de norme vectorielle sur $\mathbb{K}^{\mathtt{N}}$.

DÉMONSTRATION. On remarque que

$$A*\mathtt{X}=B\Longleftrightarrow A_1*\mathtt{X}=A_2*\mathtt{X}+B\Longleftrightarrow \mathtt{X}=A_1^{-1}*A_2*\mathtt{X}+A_1^{-1}*B.$$

Fixons ϵ dans l'intervalle ouvert $]0,1-\rho(A_1^{-1}*A_2)[$ et choisissons comme indiqué (en invoquant la proposition 1.1) une norme vectorielle sur $\mathbb{K}^{\mathtt{N}}$ telle que pour la norme matricielle induite on ait

$$\|A^{-1}*A_2\|^{(\epsilon)}\leq \rho(A_1^{-1}\cdot A_2)+\epsilon=k_\epsilon<1.$$

L'application

$$\mathtt{X}\in\mathbb{K}^{\mathtt{N}}\longmapsto A_1^{-1}*A_2*\mathtt{X}+A_1^{-1}*B$$

est alors strictement contractante avec ce choix de norme puisque $\kappa_\epsilon<1$. Le théorème du point fixe 3.1 s'applique et on obtient les conclusions voulues. Puisque toutes les normes sur $\mathbb{K}^{\mathtt{N}}$ étant équivalentes (il s'agit d'un $\mathbb{K}$-espace vectoriel de dimension finie), le fait qu'une application de $\mathbb{K}^{\mathtt{N}}$ dans lui-même soit strictement contractante pour cette norme particulière $\|\ \|^{(\epsilon)}$ (et donc que l'on ait le contrôle (3.8)) implique que l'on dispose d'un contrôle similaire pour une norme quelconque, quitte à multiplier le second membre par une constante multiplicative adéquate (dépendant bien sûr du choix de ϵ). Comme $\mathtt{X}$ est unique (unicité du point fixe), le système $A*\mathtt{X}=B$ considéré est bien de Cramer et la matrice A est forcément inversible. □

dans `Sage`) ; tel n'est pas le cas des méthodes indirectes relevant du second angle d'attaque que nous détaillons dans ce chapitre.

3.2.2. Les méthodes itératives de Jacobi et de Gauß-Seidel. Lorsque A est une matrice à entrées réelles ou complexes de taille `[N,N]` dont les coefficients diagonaux a_{jj} sont tous non nuls, on note

`DA = diagonal_matrix(A.diagonal())`

(ici suivant la syntaxe `Sage`, qu'il convient de traduire en `diag(diag(A))` sous `Scilab` ou `MATLAB`) la matrice extraite de A en ne conservant que les termes diagonaux. On note aussi `TA = tril(A)` (`tri`angular, `l`ower) la matrice extraite de A en n'en conservant que les entrées sur et sous la diagonale, suivant les instructions suivantes sous `Sage` :

```
sage:
def tril(A):
    N  = A.ncols()
    T  = Matrix([[A[j,k] for k in range(j+1)]
         + [0 for k in range(N-j-1)] for j in range(N)])
    return T
```

Notons que cette matrice `TA` se déclare de manière identique (`TA = tril(A)`) sous `Scilab` ou `MATLAB`. On dispose donc des deux écritures possibles

$$A = \mathtt{DA} - (\mathtt{DA} - A) \quad \text{ou} \quad A = \mathtt{TA} - (\mathtt{TA} - A)$$

comme écritures candidates pour $A = A_1 - A_2$ avec A_1 aisément inversible.

La première

$$A = \mathtt{DA} - (\mathtt{DA} - A) = \mathtt{DA} - \mathtt{EA} \tag{3.9}$$

soutend l'*algorithme itératif de Jacobi*[1], tandis que la seconde

$$A = \mathtt{TA} - (\mathtt{TA} - A) = \mathtt{TA} - \mathtt{FA} \tag{3.10}$$

soutend l'*algorithme itératif de Gauß*[2]*-Seidel*[3].

Exemple 3.1. La décomposition

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ 3 & 4 & -8 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & -8 \end{pmatrix} - \begin{pmatrix} 0 & -2 & -2 \\ -1 & 0 & -3 \\ -3 & -4 & 0 \end{pmatrix}$$

illustre sur un exemple de matrice de taille `[3,3]` l'approche préparatoire $A = \mathtt{DA} - \mathtt{EA}$ de Jacobi, tandis que la décomposition

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ 3 & 4 & -8 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 1 & 6 & 0 \\ 3 & 4 & -8 \end{pmatrix} - \begin{pmatrix} 0 & -2 & -2 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{pmatrix}$$

illustre sur le même exemple celle de Gauß-Seidel $A = \mathtt{TA} - \mathtt{FA}$.

1. Algébriste et géomètre allemand Carl Gustav Jacobi, 1804-1851, marqua l'essor des mathématiques au XIX$^{\text{e}}$ siècle (déterminants, fonctions elliptiques, algèbre linéaire, problèmes géométriques d'intersection,...).

2. On retrouve ici le mathématicien, astronome et philosophe allemand Carl Friedrich Gauß, 1777-1855, certainement l'un de ceux qui ont le plus contribué guider l'évolution des mathématiques tous domaines confondus (algèbre, analyse, théorie des nombres et géométrie).

3. Au nom de Gauß, se trouve ajouté celui Philipp Ludwig von Seidel, 1821-1896, élève de Jacobi, astronome, géomètre et probabiliste allemand.

Soit A une matrice à entrées dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$, de taille `[N,N]`, ayant tous ses termes diagonaux non nuls et B un vecteur de $\mathbb{K}^{\mathtt{N}}$. Les deux procédures [1] (de Jacobi et de Gauss-Seidel) s'implémentent ainsi sous `Sage` dans le cadre d'une précision de `Ndigits`, avec ici un nombre maximal d'itérations autorisé `Niter` et un test d'arrêt [2] :

```
sage:
def JACOBI(A,B,Ndigits,Niter,tol):
    N   = A.ncols(); R  = RealField(Ndigits)
    DA  = diagonal_matrix(A.diagonal())
    EA  = DA - A; IDA = DA^(-1); AA = IDA*EA
    X0  = vector([R.random_element(-tol,tol)
                  for k in range(N)])
    X   = X0
    err = 2*R(1).ulp(); iter = 0
    while err > R(1).ulp() and iter < Niter:
        Xold = X; X = AA*X + IDA*B
        ERR  = X - Xold
        err  = sqrt(ERR.conjugate()*ERR)
        iter = iter + 1
    return X,iter,err

sage:
def GAUSSSEIDEL(A,B,Ndigits,Niter,tol):
    N   = A.ncols(); R  = RealField(Ndigits)
    TA  = tril(A)
    FA  = TA - A; ITA = TA^(-1); AA = ITA*FA
    X0  = vector([R.random_element(-tol,tol)
                  for k in range(N)])
    X   = X0
    err = 2*R(1).ulp(); iter = 0
    while err > R(1).ulp() and iter < Niter:
        Xold = X; X = AA*X + ITA*B
        ERR  = X - Xold
        err  = sqrt(ERR.conjugate()*ERR)
        iter = iter + 1
    return X,iter,err
```

Les codes `Scilab` réalisant les mêmes opérations [3] sont les suivants, les entrées étant cette fois, outre le seuil d'erreur `epsilon` imposé et la tolérance `tol` préalablement

1. On ne précise pour l'instant encore rien à propos de leur convergence ou non.

2. Comme nous l'avons déjà mentionné dans ce chapitre à propos des algorithmes itératifs fondés sur l'utilisation du théorème du point fixe (théorème 3.1), la variable `tol` fixe une tolérance de taille pour générer de manière aléatoire un vecteur initial `X0` dans $\mathbb{R}^{\mathtt{N}}$, vecteur dont on décide ensuite de parcourir l'orbite sous l'action d'une application T dont on espère qu'elle sera strictement contractante. Dans les deux codes ci-dessous, la ligne 4 est à rétablir dans son intégrité.

3. On fixe cette fois un seuil d'erreur `epsilon` en place d'une précision `Ndigits`. Notons que dans le cas de divergence avec amplification exponentielle des normes des sorties `X`, `MATLAB` retourne une sortie (avec des $\pm$`NaN`) tandis que `Scilab` retourne un message d'erreur.

choisie, un tableau carré A de taille `[N,N]` (figurant la matrice A) et une matrice colonne B de taille `[N,1]` (figurant un vecteur-colonne, second membre du système linéaire à résoudre) :

```
function [X,iter,err]
         = JACOBI(A,B,epsilon,Niter,tol);
[N,N] = size(A); DA = diag(diag(A)); EA = DA - A;
IDA   = DA^(-1); AA = IDA*EA ;
X0    = 2*tol*(rand(N,1,"uniform")-1/2); X = X0;
err   = 2*epsilon; iter = 0;
while (err > epsilon) & (iter < Niter)
   Xold = X; X = AA*X + IDA*B ;
   err  = norm(X-Xold); iter = iter + 1;
end
endfunction

function [X,iter,err]
         = GAUSSSEIDEL(A,B,epsilon,Niter,tol);
[N,N] = size(A); TA = tril(A); FA = TA - A;
ITA   = TA^(-1); AA = ITA*FA;
X0    = 2*tol*(rand(N,1,"uniform")-1/2); X = X0;
err   = 2*epsilon; iter = 0;
while (err > epsilon) & (iter < Niter)
   Xold = X; X = AA*X + ITA*B ;
   err  = norm(X-Xold); iter = iter + 1;
end
endfunction
```

D'après le théorème 3.1, une condition suffisante assurant la convergence des algorithmes de Jacobi ou Gauß-Seidel lorsque le second membre B est choisi arbitrairement dans $\mathbb{K}$ est que le rayon spectral $\rho(\mathtt{DA}^{-1} * \mathtt{EA})$ (dans le cas de Jacobi) ou $\rho(\mathtt{TA}^{-1} * \mathtt{FA})$ (dans le cas de Gauß-Seidel) soit strictement inférieur à 1.

Un premier exemple important où cette condition sur le rayon spectral se trouve être validée (dans les deux cas) est celui des matrices *à diagonale dominante*.

DÉFINITION 3.1 (matrice à diagonale dominante). Soit $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$. Une matrice $A = [a_{j,k}]_{0 \leq j,k \leq \mathtt{N}-1}$ de taille `[N,N]` à entrées dans $\mathbb{K}$, où j désigne l'indice de ligne et k l'indice de colonne, est dite *à diagonale dominante* si et seulement si

$$\forall\, j = 0, ..., \mathtt{N} - 1\,, \quad |a_{j,j}| > \sum_{\substack{k=0 \\ k \neq j}}^{\mathtt{N}-1} |a_{j,k}|, \tag{3.11}$$

autrement dit, compte tenu de la définition (1.4) de la norme matricielle $\| \ \|_\infty$ sur le $\mathbb{K}$-espace vectoriel des matrices de taille `[N,N]`, si et seulement si `DA` est inversible et $\|\mathtt{DA}^{-1} * \mathtt{EA}\|_\infty < 1$.

Remarque 3.1. Si A est une matrice de taille `[N,N]` à entrées dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$ et à diagonale dominante, les entrées $a_{j,j}$ de la diagonale de A sont toutes non

nulles. Les matrices DA et TA sont dans ce cas toutes deux inversibles. De plus, leur inverse se calcule aisément : trivialement dans le premier cas (il s'agit juste de prendre la matrice diagonale dont les entrées sont les inverses des entrées de DA), facilement dans le second puisque la matrice TA est triangulaire inférieure avec entrées non nulles sur la diagonale [1].

Exemple 3.2. Les matrices

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ 3 & 4 & -8 \end{pmatrix} \qquad \begin{pmatrix} 5+2i & 3 & 2 \\ 1 & 3(1+i) & 3 \\ 3+i & 4 & -8-i \end{pmatrix}$$

sont à diagonale dominante (la première en tant que matrice à entrées réelles, la seconde en tant que matrice à entrées complexes). Les matrices

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ -5 & 4 & -8 \end{pmatrix} \qquad \begin{pmatrix} 5+2i & 3 & 2 \\ 1 & 3(1+i) & 3 \\ 3-7i & 4 & -8-i \end{pmatrix}$$

ne le sont pas (en effet, la dernière ligne pose dans les deux cas problème).

Nous pouvons énoncer la proposition suivante.

PROPOSITION 3.2 (validation des algorithmes de Jacobi ou de Gauß-Seidel pour les matrices à entrées réelles ou complexes et à diagonale dominante). *Soit A une matrice de taille* [N,N] *à entrées dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$ et à diagonale dominante. On a à la fois $\rho(\mathtt{DA}^{-1} * \mathtt{EA}) < 1$ et $\rho(\mathtt{TA}^{-1} * \mathtt{FA})$ si $A = \mathtt{DA} - \mathtt{EA} = \mathtt{TA} - \mathtt{FA}$ sont respectivement les décompositions* (3.9) *et* (3.10) *de Jacobi et de Gauß-Seidel. Dans les deux cas par conséquent la matrice A est inversible. De plus, étant donné un vecteur B arbitraire dans $\mathbb{K}^{\mathtt{N}}$, la suite $(\mathtt{X}_k)_{k\geq 0}$ initiée sur un vecteur $\mathtt{X}_0$ quelconque de $\mathbb{K}^{\mathtt{N}}$ et générée ensuite par l'algorithme itératif* (3.7) *converge vers l'unique solution dans $\mathbb{K}^{\mathtt{N}}$ du système de Cramer $A * \mathtt{X} = B$.*

DÉMONSTRATION. On voit que $\|\mathtt{DA}^{-1} * \mathtt{EA}\|_\infty < 1$, ceci résultant immédiatement de la définition 3.1. On utilise ensuite l'inégalité établie à la proposition 1.1 qui assure que l'on a les majorations $\rho(\mathtt{DA}^{-1} * \mathtt{EA}) \leq \|\mathtt{DA}^{-1} * \mathtt{EA}\|_\infty < 1$ pour prouver la majoration du rayon spectral de $\mathtt{DA}^{-1} * \mathtt{EA}$ voulue. Concernant la matrice $\mathtt{TA}^{-1} * \mathtt{FA}$, on montre directement que toute valeur propre λ (complexe) de cette matrice est de module strictement inférieur à 1, ce qui impliquera bien le fait que son rayon spectral soit aussi strictement inférieur à 1. Soit donc λ une telle valeur propre complexe de $\mathtt{TA}^{-1} * \mathtt{FA}$ et v (de coordonnées $x_0, ..., x_{\mathtt{N}-1}$ dans la base canonique de $\mathbb{C}^{\mathtt{N}}$) un vecteur propre associé ; on a les équivalences :

$$\mathtt{TA}^{-1} * \mathtt{FA}{*}\mathtt{V} = \lambda \mathtt{V} \Longleftrightarrow \mathtt{FA}{*}\mathtt{V} = \lambda\, \mathtt{TA}{*}\mathtt{V}$$

$$\Longleftrightarrow \begin{cases} -\sum_{k=j+1}^{\mathtt{N}-1} a_{j,k}\, x_k = \lambda \sum_{k=0}^{j} a_{j,k}\, x_k & \forall\, j = 0, ..., \mathtt{N}-2 \\ 0 = \lambda \sum_{k=0}^{\mathtt{N}-1} a_{\mathtt{N}-1,k}\, x_k\,. \end{cases} \tag{3.12}$$

1. Le système triangulaire inférieur TA*X=Y où Y est un vecteur de $\mathbb{K}^{\mathtt{N}}$ se résout en cascade à partir de la première équation ; on détermine les coordonnées $x_0, x_1, ...$ de X de proche en proche : $x_0 = y_0/a_{0,0}$, $x_1 = (y_1 - a_{1,0}x_0)/a_{1,1}$, etc.

Supposons un instant $|\lambda| \geq 1$. Soit j_0 tel que $|x_{j_0}| = \sup_{0 \leq k \leq \mathtt{N}-1} |x_k| > 0$. On distingue ici deux cas de figure.

— Soit $j_0 \in \{0, ..., \mathtt{N}-2\}$, et l'on a alors d'après (3.12)

$$\lambda\, a_{j_0,j_0}\, x_{j_0} = -\sum_{k=j_0+1}^{\mathtt{N}-1} a_{j_0,k}\, x_k - \lambda \sum_{k<j_0} a_{j_0,k}\, x_k,$$

ce qui implique d'après l'inégalité triangulaire (couplée avec le fait que $|\lambda| \geq 1$ et que $|x_{j_0}| \geq |x_k|$ quelque soit k), que

$$|\lambda|\,|a_{j_0,j_0}|\,|x_{j_0}| \leq \sum_{k>j_0} |a_{j_0,k}|\,|x_k| + |\lambda| \sum_{k<j_0} |a_{j_0,k}|\,|x_k| \leq |\lambda|\,|x_{j_0}| \sum_{k\neq j_0} |a_{j_0,k}|,$$

soit, en divisant par $|\lambda|\,|x_{j_0}|$,

$$|a_{j_0,j_0}| \leq \sum_{k\neq j_0} |a_{j_0,k}|.$$

— Soit $j_0 = \mathtt{N}-1$, auquel cas la dernière relation dans (3.12) qui s'écrit aussi

$$a_{\mathtt{N}-1,\mathtt{N}-1}\, x_{\mathtt{N}-1} = -\sum_{k=0}^{\mathtt{N}-1} m_{\mathtt{N}-1,k}\, x_k$$

implique, toujours en utilisant le fait que $|x_{\mathtt{N}-1}| = |x_{j_0}| = \sup_k |x_k|$, que

$$|a_{\mathtt{N}-1,\mathtt{N}-1}||x_{\mathtt{N}-1}| = |a_{\mathtt{N}-1,\mathtt{N}-1}|\,|x_{j_0}| \leq \sum_{k=0}^{\mathtt{N}-2} |a_{\mathtt{N}-1,k}|\,|x_k| \leq |x_{\mathtt{N}-1}| \sum_{k=0}^{\mathtt{N}-2} |a_{\mathtt{N}-1,k}|$$

et, par conséquent, en divisant par $|x_{\mathtt{N}-1}| = |x_{j_0}|$,

$$|a_{\mathtt{N}-1,\mathtt{N}-1}| = |a_{j_0,j_0}| \leq \sum_{k=0}^{\mathtt{N}-2} |a_{\mathtt{N}-1,k}| = \sum_{k\neq j_0} |a_{j_0,k}|.$$

Dans tous les cas, quelque soit donc la valeur de $j_0 \in \{0, ..., \mathtt{N}-1\}$, on a mis en contradiction le fait que

$$|a_{j,j}| > \sum_{k\neq j} |a_{j,k}| \qquad \forall\, j = 0, ..., \mathtt{N}-1.$$

L'hypothèse $|\lambda| \geq 1$ est donc absurde, et l'on a bien prouvé ainsi (par l'absurde) que $\rho(\mathtt{TA}^{-1} * \mathtt{FA}) < 1$. Une fois ces deux majorations strictes (par 1) des rayons spectraux acquises, les assertions qui les suivent dans l'énoncé de la proposition 3.2 (à savoir que A soit inversible et que les algorithmes de Jacobi ou de Gauß-Seidel convergent tous les deux) proviennent de l'application dans ce cas de la proposition 3.1. □

La proposition 3.2 propose donc une condition suffisante (à savoir le fait que la matrice carrée de taille `[N,N]` soit à diagonale dominante) pour que les algorithmes de Jacobi ou de Gauß-Seidel convergent lorsqu'il s'agit de calculer *via* l'une ou l'autre de ces deux méthodes itératives une solution approchée du sytème d'équations linéaires $A * \mathtt{X} = B$ ($B \in \mathbb{K}^{\mathtt{N}}$, $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$), pourvu que tous les termes diagonaux de la matrice A soient non nuls. Lorsque l'on se trouve comme ici dans la configuration où l'on peut affirmer que les deux méthodes itératives (Jacobi et Gauß-Seidel) convergent, il est en général préférable d'exploiter la méthode de Gauß-Seidel : en effet, la matrice `FA` (impliquée

dans la définition de $\mathtt{AA} = \mathtt{TA}^{-1} * \mathtt{FA}$) se trouve être triangulaire supérieure sans termes diagonaux, donc *a priori* beaucoup plus « creuse » que ne l'est la matrice en général quasiment « pleine » $\mathtt{EA}$; les calculs régis par l'itération (3.7) s'en trouvent facilités en ce qui concerne leur coût. On constate d'ailleurs si l'on effectue quelques tests que l'algorithme de Gauß-Seidel converge plus rapidement. Par exemple, sous `Sage`, avec des matrices générées aléatoirement [1] :

```
sage:
Ndigits   = 100; R = RealField(Ndigits); N = 30
mintaille = -1; maxtaille = 1
mindiag   = 15; maxdiag = 20; tailleB = 5
A   = Matrix(R,[[R.random_element(mintaille,maxtaille)
          for k in range(N)] for j in range(N)])
D   = diagonal_matrix(R,[R.random_element(mindiag,maxdiag)
                      for k in range(N)])
A   = A + D;
B   = vector(R, [R.random_element(-tailleB,tailleB)
          for k in range(N)])
sage:
XJ  = JACOBI(A,B,Ndigits,100,10); ERR = XJ[0] - A^(-1)*B;
XJ[1]; sqrt(ERR*ERR)
ans:
43  2.109034800633160260436549910 8e-30

sage:
XGS  = GAUSSSEIDEL(A,B,Ndigits,100,10)
ERR  = XGS[0] - A^(-1)*B; XGS[1]; sqrt(ERR*ERR)
ans:
30  1.9933773504185373940582472513e-30
```

La méthode de Gauß-Seidel est aussi opérationnelle lorsque la matrice A (à entrées réelles ou complexes) se trouve être la matrice d'une forme quadratique `PHI` (voir la section 2.1) définie positive [2] sur $\mathbb{R}^{\mathtt{N}}$ dans le cas où il s'agit d'une matrice à entrées réelles ou la matrice d'une forme hermitienne `PHI` (voir la section 2.2) définie positive sur $\mathbb{C}^{\mathtt{N}}$ lorsqu'il s'agit d'une matrice à entrées complexes. On rappelle que dans ces deux cas, on a

$$a_{j,k} = e_j * A * e_k, \qquad 0 \leq j, k \leq \mathtt{N} - 1$$

si $\{e_0, ..., e_{\mathtt{N}-1}\}$ désigne la base canonique de $\mathbb{K}^{\mathtt{N}}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) ; en particulier, pour ce qui est des termes diagonaux, on a $a_{j,j} = e_j * A * e_j = \mathtt{PHI}(e_j)$ et par conséquent $a_{j,j} \in]0, +\infty[$ pour tout $j = 0, ..., \mathtt{N} - 1$ dans ce cas puisque `PHI` est supposée définie positive. Voici le résultat dont on dispose dans un tel cadre.

---

1. On profite ici de ces tests pour comparer (en affichant, avec le nombre d'itérations réellement consommées, la norme $\| \ \|_2$ du vecteur `ERR`) le résultat obtenu par la méthode itérative avec celui qu'aurait donné une méthode directe (vraisemblablement ici la méthode du pivot de Gauß) de résolution du système $A * \mathtt{X} = B$.

2. C'est-à-dire telle que $\mathtt{PHI} \geq 0$ et $\mathtt{PHI(X)} = 0$ si et seulement si $\mathtt{X} = 0$; ceci vaut aussi dans le cadre complexe.

PROPOSITION 3.3 (décomposition de Gauß-Seidel pour une matrice de forme hermitienne PHI définie positive). *Soit A une matrice symétrique réelle ou hermitienne définie positive de taille* [N,N] *et* $A =$ TA$-$FA *sa décomposition de Gauß-Seidel* (3.10). *On a* $\rho(\mathtt{TA}^{-1} * \mathtt{FA}) < 1$.

DÉMONSTRATION. On écrit $A =$ DA+T+T', où T=-FA est une matrice triangulaire supérieure (avec diagonale nulle) et

$$\mathtt{T'=(T.conjugate()).transpose()}.$$

La matrice DA + T' est une matrice triangulaire inférieure inversible que l'on peut factoriser ainsi en introduisant la matrice diagonale DDA à entrées strictement positives dont le carré est la matrice diagonale DA et l'inverse est noté IDDA :

$$\begin{aligned}\mathtt{DA + T'} &= \mathtt{DDA * IDDA * (DA + T') * IDDA * DDA}\\ &= \mathtt{DDA} * \big(\mathtt{matrix.identity(N) + IDDA * T' * IDDA}\big) * \mathtt{DDA}.\end{aligned}$$

Les valeurs propres de la matrice

$$\begin{aligned}\mathtt{AA} &= \mathtt{TA}^{-1} * \mathtt{FA} = -\big(\mathtt{DA+T'}\big)^{-1} * \mathtt{T}\\ &= -\mathtt{IDDA} * \big(\mathtt{matrix.identity(N) + IDDA*T'*IDDA}\big)^{-1} * \mathtt{IDDA} * \mathtt{T}\end{aligned} \tag{3.13}$$

sont les mêmes que celles de la matrice $\mathtt{IDDA}^{-1} * \mathtt{AA} * \mathtt{IDDA}$, donc, compte-tenu de (3.13), les opposées de celles de la matrice

$$\mathtt{B} = \big(\mathtt{matrix.identity(N)} + \mathtt{TT}\big)^{-1} * \mathtt{TT'}$$

où

$$\mathtt{TT} := \mathtt{IDDA * T'* IDDA}.$$

Si v est un vecteur propre (dans $\mathbb{C}^{\mathtt{N}}$) de norme 1 associé à une valeur propre complexe λ de la matrice B, on a

$$\big(\mathtt{matrix.identity(N) + TT'}\big)^{-1} * \mathtt{TT' * V} = \lambda\, \mathtt{v}$$

soit

$$\mathtt{TT'*V} = \lambda \mathtt{v} + \lambda \mathtt{TT} * \mathtt{v}.$$

On a par conséquent, en prenant le produit scalaire à gauche avec V et en utilisant toujours les notations empruntées à Scilab ou MATLAB pour figurer la transconjuguaison :

$$\mathtt{V'*TT'*V} = \lambda\, \|\mathtt{V}\|^2 + \lambda\, \mathtt{V'*TT*V}$$

et, par conséquent, puisque v est unitaire,

$$|\mathtt{V'*TT'*V}|^2 = |\lambda|^2 \times \big|1 + \mathtt{V'*TT*V}\big|^2. \tag{3.14}$$

Mais, comme la matrice A, la matrice

$$\mathtt{IDDA} * A * \mathtt{IDDA} = \mathtt{matrix.identity(N)} + \mathtt{TT} + \mathtt{TT'}$$

est la matrice d'une forme hermitienne définie positive, ce qui implique que pour tout vecteur X de $\mathbb{K}^{\mathtt{N}} \setminus \{(0, ..., 0)\}$ on ait

$$\|\mathtt{X}\|^2 + 2\,\mathrm{Re}\,(\mathtt{X' * TT * X}) > 0. \tag{3.15}$$

Ceci est en particulier vrai pour `X` $=$ `v`. Si l'on pose `V'*TT*V` $= x + iy$, on déduit de (3.14) que

$$x^2 + y^2 = |\lambda|^2 \, (1 + 2x + x^2 + y^2) > |\lambda|^2 \, (x^2 + y^2 + \eta)$$

avec $\eta = (1 + 2x)/2 > 0$ (d'après 3.15). Il en résulte bien $|\lambda|^2 < 1$, donc $|\lambda| < 1$, ce qu'il fallait prouver pour conclure. □

Attention : la convergence de la méthode de Gauß-Seidel sous les hypothèses de la proposition 3.3 peut s'avérer toutefois très lente, surtout si la suite des valeurs propres de A ne « décroche » pas assez vite de sa valeur maximale.

Exemple 3.3 (calcul de projection orthogonale)**.** Soient $\mathtt{v}_0$,..., $\mathtt{v}_{m-1}$ une liste de m vecteurs de $\mathbb{K}^{\mathtt{N}}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) formant un système libre. Pour calculer la projection orthogonale d'un vecteur `X` de $\mathbb{K}^{\mathtt{N}}$ sur le $\mathbb{K}$-sous-espace vectoriel de dimension m engendré par les vecteurs $\mathtt{v}_j$ pour $j = 0, ..., m-1$, on peut envisager deux manières de procéder.

— La première consiste à orthonormaliser le système libre $\{\mathtt{v}_0, ..., \mathtt{v}_{m-1}\}$ de $\mathbb{K}^{\mathtt{N}}$ pour le produit scalaire canonique dans cet espace. Ceci se conduit en utilisant la routine `GRAMSCHMIDT` proposée à la section 2.3.1. Une fois le système orthonormé $\{\mathtt{e}_0, ..., \mathtt{e}_{m-1}\}$ connu, on a (avec la syntaxe de `Sage` et en convenant toujours que `X'` $=$ `X.conjugate()`)

$$\mathrm{Proj}_{\mathrm{vect}(\mathtt{v}_0,...,\mathtt{v}_{m-1})}[\mathtt{X}] = \sum_{j=0}^{m-1} \langle \mathtt{X} \, , \, \mathtt{e}_j \rangle \, \mathtt{e}_j = \sum_{j=0}^{m-1} (\mathtt{e}'_j * \mathtt{X}) * \mathtt{e}_j$$

puisque la coordonnée de cette projection orthogonale le long du vecteur $\mathtt{e}_j$ est égale au produit scalaire $\langle \mathtt{X}, \mathtt{e}_j \rangle = \mathtt{e}'_j * \mathtt{X}$.

— La seconde consiste à calculer explicitement la matrice de Gram du système $\{\mathtt{v}_0, ..., \mathtt{v}_{m-1}\}$, c'est-à-dire la matrice dont les entrées sont les produits scalaires $\langle \mathtt{v}_j \, , \, \mathtt{v}_k \rangle$, $0 \leq j, k \leq m-1$. Cette matrice A (de taille `[m,m]`) est la matrice d'une forme hermitienne définie positive. Trouver la projection orthogonale de `X` exprimée cette fois sous la forme

$$\mathrm{Proj}_{\mathrm{vect}(\mathtt{v}_0,...,\mathtt{v}_{m-1})}[\mathtt{X}] = \sum_{j=0}^{m-1} y_j \, \mathtt{v}_j$$

revient à résoudre le système de Cramer $A * \mathtt{Y} = B$, où B désigne le vecteur dont les coordonnées sont les $\langle \mathtt{X}, \mathtt{v}_j \rangle = \mathtt{v}'_j * \mathtt{X}$, $j = 0, ..., m-1$. Le calcul de `Y` peut cette fois être réalisé de manière itérative par la méthode de Gauß-Seidel (convergente en vertu de la proposition 3.3).

Bien souvent, l'algorithme d'orthonormalisation de Gram-Schmidt peut s'avérer numériquement très instable, par exemple lorsque l'un des vecteurs $\mathtt{v}_{j_0}$ de la liste proposée ($j_0 \in \{1, ..., m-1\}$) a une projection orthogonale de norme très petite sur l'orthogonal dans $\mathbb{K}^{\mathtt{N}}$ du $\mathbb{K}$-sous-espace vectoriel engendré par $\mathtt{v}_0, ..., \mathtt{v}_{j_0-1}$, ce qui sous-entend une « presque division par zéro » dans la mise en œuvre de cet algorithme. La méthode de Gauß-Seidel (seconde approche) doit dans ce cas être privilégiée par rapport à celle fondant la première approche, c'est-à-dire l'algorithme d'orthonormalisation de Gram-Schmidt, susceptible de se révéler alors instable numériquement. Voici l'implémentation de ces deux calculs de projection orthogonale sous `Sage` suivant le

procédé d'orthonormalisation (`PROJECTIONORTH1`), puis l'algorithme de Gauß-Seidel (`PROJECTIONORTH2`). Les entrées sont ici une liste `L` de vecteurs $\mathtt{v}_0, ..., \mathtt{v}_{m-1}$ formant un système libre de $\mathbb{K}^{\mathtt{N}}$; le vecteur `X` est le vecteur à projeter ; on a fixé de plus un nombre maximal d'itérations autorisées `Niter` ainsi qu'une tolérance `tol` dans la routine `PROJECTIONORTH2` faisant appel à l'algorithme de Gauß-Seidel.

```
sage:
def PROJECTIONORTH1(L,X,Ndigits):
    if L[0][0] in RealField(Ndigits):
       corps = RealField(Ndigits)
    else:
       corps = ComplexField(Ndigits)
    Lo = GRAMSCHMIDT(corps,L)
    V  = sum((Lo[0][k].conjugate()*X)*Lo[0][k]
              for k in range(N1))
    return V

sage:
def PROJECTIONORTH2(L,X,Ndigits,Niter,tol):
    A  =Matrix([[L[j].conjugate()*L[k] for k
              in range(N1)] for j in range(N1)])
    B  =vector([L[j].conjugate()*X for j in range(N1)])
    GS = GAUSSSEIDEL(A,B,Ndigits,Niter,tol)
    V  = sum(GS[0][k]*L[k] for k in range(N1))
    return V, GS[1], GS[2]
```

3.3. Calcul (itératif) du rayon spectral et méthode de la puissance

Une autre application importante des méthodes itératives en algèbre linéaire (sur le corps $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) est le calcul approché du rayon spectral d'une matrice de taille $[N, N]$. On rappelle que le rayon spectral d'une matrice carrée à entrées complexes est défini comme le rayon du plus petit disque fermé du plan complexe de centre l'origine qui contienne tout le spectre complexe de la matrice, c'est-à-dire toutes les valeurs propres (dans $\mathbb{C}$) de cette matrice. On a vu à la section précédente combien une information sur le rayon spectral d'une certaine matrice (en l'occurrence le fait qu'il soit strictement inférieur à 1) constituait une information essentielle pour permettre de valider la convergence de certains algorithmes itératifs (Jacobi, Gauß-Seidel par exemple).

Nous disposons en effet du résultat (algorithmique) suivant :

Proposition 3.4 (algorithme itératif pour le calcul approché du rayon spectral). *Soit A une matrice à coefficients dans $\mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{K} = \mathbb{C}$), diagonalisable*[1] *sur $\mathbb{C}$,*

1. C'est le cas, presque sûrement, pour une matrice dont les coefficients sont pris au hasard (suivant une loi uniforme) dans un segment $[a, b]$ de $\mathbb{R}$ ou un pavé $[a, b] \times [c, d]$ du plan complexe. Pour qu'il en soit ainsi, il faut que les coefficients du polynôme caractéristique (donc les entrées de la matrice A) soient liés par une certaine relation de dépendance algébrique, évènement se produisant avec une probabilité nulle lorsque la matrice avec laquelle on travaille est générée aléatoirement.

telle que les valeurs propres (distinctes ou confondues) aient des modules s'organisant comme suit[1]

$$|\lambda_0| > |\lambda_\mu| \geq |\lambda_{\mu+1}| \geq ... \geq |\lambda_{\mathtt{N}-1}| \geq 0$$

(avec $\mu \in \mathbb{N}^*$ *et* $\lambda_0 = \lambda_1 = \cdots = \lambda_{\mu-1}$*). Soit* $(\mathtt{V}_0, ..., \mathtt{V}_{\mathtt{N}-1})$ *une base de* $\mathbb{C}^{\mathtt{N}}$ *constituée de vecteurs propres pour* A *telle que* $\mathtt{V}_0, ..., \mathtt{V}_{\mu-1}$ *soit une base du sous espace propre associé à la valeur propre de module maximal* λ_0*. On se donne un vecteur*

$$\mathtt{X}_0 = x_{0,0}\mathtt{V}_0 + \cdots + x_{0,\mathtt{N}-1}\mathtt{V}_{\mathtt{N}-1}$$

de $\mathbb{K}^{\mathtt{N}}$ *tel que l'un au moins des* $x_{0,k}$*,* $k = 0, ..., \mu - 1$*, soit non nul*[2]*. Sous ces hypothèses, l'algorithme itératif initié à* $\mathtt{X}_0$ *et régi ensuite par*

$$\mathtt{X}_{k+1} = \frac{A * \mathtt{X}_k}{\|A * \mathtt{X}_k\|}\,,\ k \geq 0$$

(une norme sur $\mathbb{K}^{\mathtt{N}}$ *ayant été arbitrairement choisie) génère une suite de vecteurs* $(\mathtt{X}_k)_{k\geq 0}$ *telle que*

$$\lim_{k\to+\infty} \|A \cdot \mathtt{X}_k\| = |\lambda_0|$$

et fournit donc un moyen numérique d'approcher le rayon spectral de A*. La vitesse de convergence est de plus exponentielle*[3]*.*

Remarque 3.2. Le choix d'une norme doit être fait au préalable ; sous `Scilab` ou `MATLAB`, la norme que l'on choisit en priorité est la norme euclidienne (c'est-à-dire la norme $\|\ \|_2$, de fait norme par défaut) mais l'on pourrait prendre aussi n'importe laquelle des normes $\|\ \|_p$, $p \in [1, \infty]$. La même remarque vaut pour ce qui concerne l'implémentation de cette routine sous `Sage`.

Démonstration. Montrons d'abord par récurrence sur k que, pour tout entier $k \geq 0$,

$$\mathtt{X}_k = \frac{A^k * \mathtt{X}_0}{\|A^k * \mathtt{X}_0\|}\,.$$

Ceci est vrai pour $k = 0$ par définition de $\mathtt{X}_0$ et l'on a, pour $k \geq 0$, si l'on admet le résultat au cran k,

$$\mathtt{X}_{k+1} = \frac{A * \mathtt{X}_k}{\|A * \mathtt{X}_k\|} = A * \Big(\frac{A^k * \mathtt{X}_0}{\|A^k * \mathtt{X}_0\|}\Big) \times \Big(\frac{\|A^{k+1} * \mathtt{X}_0\|}{\|A^k * \mathtt{X}_0\|}\Big)^{-1} = \frac{A^{k+1} * \mathtt{X}_0}{\|A^{k+1} * \mathtt{X}_0\|}\,,$$

1. Il n'y a donc qu'une seule valeur propre de plus grand module, c'est le cas par exemple pour une matrice dont toutes les entrées sont positives (il s'agit là d'un résultat important dû aux algébristes allemands Oskar Perron, 1880-1975, et Ferdinand Frobenius, 1849-1917, et que nous avons déjà mentionné, voir par exemple la section V.3 de [**MathAppL3**]), ou d'une matrice dont toutes les valeurs propres sont réelles (par exemple une matrice symétrique réelle ou une matrice hermitienne complexe), pourvu qu'il n'y ait pas deux valeurs propres de signe opposé et toutes deux de valeur absolue maximale.

2. C'est encore le cas d'un $\mathtt{X}_0$ pris au hasard (de manière uniforme) dans $\mathbb{K}^{\mathtt{N}}$, ce avec une probabilité égale à 1.

3. Cependant la convergente s'avère lente, ce d'autant plus que le quotient $|\lambda_\mu|/|\lambda_0|$ se trouve être proche de 1. Néanmoins, cette méthode se trouve exploitée lorsque l'on a à traiter des matrices A de très grande taille (à entrées positives) telles les matrices d'adjacence pondérées impliquées dans l'algorithme `Pagerank` (voir par exemple la section 4.2 de [**CalcLog**]).

ce qui prouve le résultat au cran $k+1$. Or

$$\begin{aligned} A^k * \mathtt{X}_0 &= \sum_{j=0}^{\mathtt{N}-1} \lambda_j^k x_{0,j} \mathtt{V}_j = \lambda_0^k \Big[\Big(\sum_{j=0}^{\mu-1} x_{0,j} \mathtt{V}_j \Big) + \sum_{j=\mu}^{\mathtt{N}-1} \Big(\frac{\lambda_j}{\lambda_0} \Big)^k x_{0,j} \mathtt{V}_j \Big] \\ &= \lambda_0^k \Big(\sum_{j=0}^{\mu-1} x_{0,j} \mathtt{V}_j + \vec{\epsilon}_k \Big) \end{aligned}$$

avec

$$\|\vec{\epsilon}_k\| < \|\mathtt{X}_0\| (|\lambda_\mu|/|\lambda_0|)^k \, .$$

On a donc

$$|\lambda_0|^k \Big\| \sum_{j=0}^{\mu-1} x_j \mathtt{V}_j \Big\| (1-\eta_k) \leq \|A^k * \mathtt{X}_0\| \leq |\lambda_0|^k \Big\| \sum_{j=0}^{\mu-1} x_{0,j} \mathtt{V}_j \Big\| (1+\eta_k) \tag{3.16}$$

avec

$$|\eta_k| < (|\lambda_\mu|/|\lambda_0|)^k \frac{\|\mathtt{X}_0\|}{\Big\| \sum_{j=0}^{\mu-1} x_{0,j} \mathtt{V}_j \Big\|} \, .$$

On a d'autre part

$$A * \mathtt{X}_k = \frac{\lambda_0^k \big(\lambda_0 \sum_{j=0}^{\mu-1} x_{0,j} \mathtt{V}_j + A * \vec{\epsilon}_k \big)}{\|A^k * \mathtt{X}_0\|}$$

et, en prenant les normes

$$\frac{|\lambda_0|^{k+1} \Big\| \sum_{j=0}^{\mu-1} x_{0,j} e_j \Big\| (1-\tilde{\eta}_k)}{\|A^k * \mathtt{X}_0\|} \leq \|A * \mathtt{X}_k\| \leq \frac{|\lambda_0|^{k+1} \Big\| \sum_{j=0}^{\mu-1} x_{0,j} \cdot \mathtt{V}_j \Big\| (1+\tilde{\eta}_k)}{\|A^k * \mathtt{X}_0\|} \tag{3.17}$$

avec

$$\tilde{\eta}_k \leq (|\lambda_\mu|/|\lambda_0|)^k \|A\| \frac{\|\mathtt{X}_0\|}{|\lambda_0| \Big\| \sum_{j=0}^{\mu-1} x_{0,j} \mathtt{V}_j \Big\|} \, .$$

On achève donc la preuve en combinant les encadrements (3.16) et (3.17). Comme $(|\lambda_\mu|/|\lambda_0|)^k$, la décroissance vers 0 de η_k et $\tilde{\eta}_k$ est en $e^{-\rho k}$ avec $\rho > 0$ (puisque $|\lambda_\mu| < |\lambda_0|$) et l'on a bien une vitesse exponentielle de convergence de $\|A * \mathtt{X}_k\|$ vers $|\lambda_0|$. □

Remarque 3.3. Si la matrice A est à coefficients réels et admet au moins une valeur propre λ de module maximal qui, elle, n'est pas réelle, la condition d'application de cet algorithme itératif ne saurait être remplie (puisque la matrice A possède alors au moins deux valeurs propres complexes de module maximal distinctes, à savoir λ et sa conjuguée $\bar{\lambda}$). Ainsi, lorsque A se trouve être une matrice à entrées réelles, la démarche itérative présentée ici ne saurait fonctionner que si la seule valeur propre de A de module maximal est réelle. On en voit donc les limites.

Le code correspondant à l'implémentation de cet algorithme est rédigé ainsi sous `Sage` (veiller à rétablir la ligne 2 dans son intégrité) :

```
sage:
def RAYONSPECTRAL(A,N,Ndigits,tol,Niter):
    R   = RealField(Ndigits)
    X0  = vector([R.random_element(-tol,tol)
          for k in range(N)])
    err = 2*R(1).ulp(); iter = 0; r = R(1); X = X0
    while err > R(1).ulp() and iter < Niter:
        rold = r; X = A*X; r = sqrt(X.conjugate()*X)
        X    = X/r; err = abs(r-rold); iter = iter + 1
    return r,iter
```

Nous avons ici incorporé un test d'arrêt (lorsque l'on travaille avec `Ndigits` bits de précision en réel). Le vecteur `V0` est généré aléatoirement (à coordonnées réelles indépendantes et suivant une loi uniforme sur $[-\texttt{tol}, \texttt{tol}]$). Si $|\lambda_\mu|/|\lambda_0|$ se trouve être proche de 1 (tout en lui étant strictement inférieur), il convient malheureusement de choisir `Niter` très grand pour espérer un résultat convenable. On peut confronter ce résultat avec le calcul des racines du polynôme caractéristique *via* la méthode de Newton (tel qu'il est opéré dans `Sage`).

Il est possible de modifier ce code pour qu'il retourne, sous l'hypothèse additionnelle que l'unique valeur propre λ_0 de module maximal soit une valeur propre simple (cette hypothèse supplémentaire est malheureusement ici essentielle et on ne saurait s'en affranchir) un vecteur propre unitaire (c'est-à-dire de norme euclidienne 1) associé à la valeur propre λ_0. En voici l'implémentation `Sage`. Le test d'arrêt que nous avons mis ici est calé sur l'indicateur qu'est la norme de `A*Vp-lambd*Vp`.

```
sage:
def METHODEPUISSANCE(A,N,Ndigits,tol,Niter):
         R     = RealField(Ndigits)
         X0    = vector([R.random_element(-tol,tol)
                 for k in range(N)])
         X0    = X0/sqrt(X0.conjugate()*X0); Vp  = X0
         lambd = X0.conjugate()*A*X0
         err   = 2*R(4*abs(lambd)).ulp(); iter =0
         while err>R(4*abs(lambd)).ulp() and iter<Niter:
             Vpaux = A*Vp
             Vp    = Vpaux/sqrt(Vpaux.conjugate()*Vpaux)
             lambd = Vp.conjugate()*A*Vp
             ERR   = A*Vp - lambd*Vp
             err   = sqrt(ERR.conjugate()*ERR)
             iter  = iter + 1
         return lambd,Vp,iter, err
```

(il convient ici de restaurer la ligne 2 de la procédure dans son intégrité). Ce code n'est opérationnel que si λ_0 est seule valeur propre de module maximal, avec de plus $|\lambda_1| < |\lambda_0|$ (la valeur propre λ_0 doit donc être simple). Le code ainsi réalisé correspond à la mise en œuvre de la *méthode de la puissance*.

Les codes `Scilab` correspondant aux deux codes `RAYONSPECTRAL` et `METHODEPUISSANCE` élaborés sous `Sage` précédemment sont les deux codes `.sci` suivants :

```
function [r,iter]
      = RAYONSPECTRAL(A,tol,epsilon,Niter);
[N,N] = size(A);
X0    = 2*tol*(rand(N,1,"uniform")-1/2);
err   = 2*epsilon; iter = 0; r = 1; X = X0;
while (err>epsilon) & (iter < Niter)
rold  = r; X = A*X; r=norm(X); X = X/r;
err   = abs(r-rold); iter = iter+1;
end
endfunction

function [lambd,Vp,iter,err]
       = METHODEPUISSANCE(A,epsilon,tol,Niter)
[N,N]  = size(A);
X0     = 2*tol*(rand(N,1,"uniform")-1/2);
X0     = X0/norm(X0); Vp = X0; lambd = X0'*A*X0;
err    = 2*epsilon ; iter = 0;
while (err > epsilon) & iter < Niter
Vpaux  = A*Vp; Vp = Vpaux/norm(Vpaux);
lambd  = Vp'*A*Vp;
err    = norm(A*Vp-lambd*Vp);
iter   = iter + 1;
end
endfunction
```

On a fixé cette fois au préalable (dans les trois cas de figure) un seuil d'erreur `epsilon` ; la syntaxe des codes est identique à celle de leurs analogues sous `Sage`. Ces routines sont disponibles dans le répertoire `ScilabMATLAB - CHAP4` sur le site `web` dédié. Notons que le calcul du rayon spectral d'une matrice carrée A à entrées complexes (*via* le calcul préalable de ses valeurs propres) est retourné sous `Scilab` ou `MATLAB` par

```
--> r = max(abs(spec(A)));
>>  r = max(abs(eig(A)));
```

On pourra ainsi confronter les résultats retournés par les deux codes itératifs construits ici (ou leurs clones sous `MATLAB`) avec ceux retournés par l'une ou l'autre de ces instructions.

Cette méthode de la puissance peut être combinée avec le principe de déflation (lemme (2.1)) pour donner la recherche de proche en proche des valeurs propres lorsqu'elles sont toutes simples et de module distinct. Voici par exemple une procédure permettant de trigonaliser une matrice complexe (procédure de Cayley Hamilton). Cette procédure fournit en sortie deux matrices `P` et `T`, la première inversible, la seconde triangulaire supérieure, telles que `A = P*T*P`$^{-1}$ si `A` est la matrice d'entrées.

```
sage:
def TRIGONALISATION(A,N,Ndigits,tol,Niter):
    R = RealField(Ndigits); C = ComplexField(Ndigits)
    I = identity_matrix(N)
# phase d'initialisation (il s'agit d'un code recursif)
    if N == 1:
        return Matrix([[R(1)]]),A
    else:
# calcul de la valeur propre de module dominant
# et d'un vecteur propre associe
        RAY   = METHODEPUISSANCE(A,N,Ndigits,tol,Niter)
        lambd = RAY[0]; Vp = RAY[1]
# completion du vecteur propre en une base et
# formation de la matrice de passage P correspondante
        LVp   = Vp.list()
        LLVp = [abs(LVp[k]) for k in range(N)]
        mxLVp = max(LLVp); ixL  = LLVp.index(mxLVp)
        LI    = [I[k] for k in range(N)]
        LI.remove(LI[ixL])
        LP    = [Vp] + LI; P = Matrix(C,LP).transpose()
# mise en place de la recursivite
        AA    = P^(-1)*A*P
        A0    = AA.matrix_from_rows([k+1
                    for k in range(N-1)])
        A0    = A0.matrix_from_columns([k+1
                    for k in range(N-1)])
# appel a la recursivite
        TRIG0 = TRIGONALISATION(A0,N-1,Ndigits,tol,Niter)
# phase finale (nouvelle matrice de passage)
        LPP   = [I[0].list()]
        for j in range(N-1):
            LPP = LPP + [[0] + TRIG0[0][j].list()]
        PP    = Matrix(C,LPP)
        T     = PP^(-1)*AA*PP; P = P*PP
        return P,T
```

Bibliographie

[CalcLog] Calcul Scientifique et Calcul Symbolique, Éléments de cours illustrés par des TP guidés sous les environnements `Maple`, `MATLAB` ou `Scilab`, `Sage` sous `Python`, collection Références Sciences, éditions Ellipses, Paris, 2015.

[MathAppL3] Mathématiques Appliquées L3, J.A. Weil & A. Yger, ed., Pearson Education, Paris, 2009.