D.M. de Mathématiques # 4
à rendre le Jeudi 06 Novembre 2018

Lycée Chaptal
BCPST2 A

La qualité de la rédaction, la clarté et la précision des raisonnements, entreront pour une part importante dans l'appréciation de la copie. Les abréviations, sigles ou phrases nominales sont à proscrire. La numérotation des exercices (et des questions) doit être respectée et mise en évidence. Les résultats doivent être encadrés proprement. Chaque candidat est responsable de la vérification de son sujet d'épreuve : pagination et impression de chaque page. Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il convient de le signaler sur la copie et de poursuivre la composition en expliquant les raisons des initiatives qui ont été prises.
Le devoir peut être fait à deux, les deux écritures doivent ***impérativement*** *apparaître sur la copie et en parts égales.*
On rappelle qu'il vaut mieux, à titre exceptionnel, ne pas rendre un devoir maison plutôt que de recopier une correction trouvée sur internet.

Exercice 1 Soit f l'application linéaire de $\mathbf{R}^3$ dans lui-même dont la matrice M dans la base canonique $\mathcal{B}_c$ de $\mathbf{R}^3$ est donnée par :

$$\mathrm{M} = \begin{pmatrix} 1 & -2 & 2 \\ -1 & 0 & 1 \\ 1 & -1 & 2 \end{pmatrix}.$$

□ **1**— Soient $u_1 = (1,1,0)$, $u_2 = (0,1,1)$ et $u_3 = (1,0,1)$.
- **1.1.** Montrer que $\mathcal{B} = (u_1, u_2, u_3)$ est une base de $\mathbf{R}^3$.
- **1.2.** Déterminer la matrice N de f dans la base $\mathcal{B}$.

□ **2**— Soit $\mathrm{P} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$. Justifier que P est inversible et calculer son inverse.

□ **3**— Montrer sans calcul que $\mathrm{N} = \mathrm{P}^{-1}\mathrm{MP}$.

□ **4**— Calculer N^n pour tout n, puis M^n.

➥ **SOLUTION de l'Exercice 1.**

□ **1**—
- **1.1.** On calcule par exemple le rang de la famille : $\mathrm{Rg}\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \mathrm{Rg}\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix}$ en faisant $L_2 \leftarrow L_2 - L_1$. Cette dernière matrice est de rang $1 + \mathrm{Rg}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$. Or, comme $\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix} = 2 \neq 0$, la matrice $\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$ est donc de rang 2. Nous avons donc au final une matrice de rang trois. Conclusion : la famille $\mathcal{B}$ est une base.
- **1.2.** On a $f(u_1) = -u_1$, $f(u_2) = u_2$ et $f(u_3) = u_3$. D'où : $\underset{\mathcal{B},\mathcal{B}}{\mathcal{M}\mathrm{at}}(f) = \mathrm{Diag}(-1,1,3)$.

□ **2**— On considère la matrice de passe $\mathrm{P}^{\mathcal{B}_c \to \mathcal{B}} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$. L'algorithme du pivot de Gauß nous fournit l'inversibilité (trois pivots non nuls) et également l'inverse de $\mathrm{P}^{\mathcal{B}_c \to \mathcal{B}}$: $\mathrm{P}^{\mathcal{B}_c \to \mathcal{B}} = \frac{1}{2}\begin{pmatrix} 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \end{pmatrix}$.

□ **3**— C'est simplement une formule de changement de base entre $\mathcal{B}_c$ et $\mathcal{B}$.

□ **4**— Puisque N est diagonale, on a : $\mathrm{N}^n = \begin{pmatrix} (-1)^n & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3^n \end{pmatrix}$. On en déduit alors $\mathrm{M}^n = \mathrm{PN}^n\mathrm{P}^{-1}$ d'où après calculs :

$$\frac{1}{2}\begin{pmatrix} (-1)^n + 3^n & (-1)^n - 3^n & (-1)^{n+1} + 3^n \\ (-1)^n - 1 & (-1)^n + 1 & (-1)^{n+1} + 1 \\ -1 + 3^n & 1 - 3^n & 1 + 3^n \end{pmatrix}.$$

■

Exercice 2 **Quelques endomorphismes nilpotents.** Dans cet exercice la notation $\mathbf{K}$ désignera $\mathbf{R}$ ou $\mathbf{C}$. Si E est un espace vectoriel et $f \in \mathcal{L}(E)$, on note f^k l'endomorphisme

$$f^k = \underbrace{f \circ f \ldots \circ f}_{k \text{ fois}}.$$

Un endomorphisme de E est dit *nilpotent* s'il existe un entier naturel m tel que $f^m = 0_{\mathcal{L}(E)}$.

□ **1**— Soit f l'application de $\mathbf{K}^n$ dans $\mathbf{K}^n$ qui a tout n-uplet $(x_1, x_2, \ldots, x_{n-1}, x_n) \in \mathbf{K}^n$ associe le n-uplet $(0, x_1, x_2, \ldots, x_{n-1}) \in \mathbf{K}^n$.

- **1.1.** Vérifier que $f \in \mathcal{L}(\mathbf{K}^n)$. Déterminer son noyau et son image.
- **1.2.** Montrer que f est nilpotent.

□ **2**— On définit $\Delta : \left| \begin{array}{ccc} \mathbf{K}[X] & \longrightarrow & \mathbf{K}[X] \\ P & \longmapsto & P(X+1) - P(X) \end{array} \right.$.

- **2.1.** Quelle relation y-a-t-il entre le degré de P et celui de $\Delta(P)$?
- **2.2.** Montrer que $\Delta|_{\mathbf{K}_n[X]}$ est un endomorphisme de $\mathbf{K}_n[X]$. Nous le noterons Δ_n dans la suite.
- **2.3.** Déterminer le noyau de Δ_n puis son image.
- **2.4.** Montrer que Δ_n est nilpotent.

□ **3**— Soit E un $\mathbf{K}$-espace vectoriel de dimension finie et $f \in \mathcal{L}(E, E)$ nilpotent. Soit m le plus petit entier strictement positif tel que : $f^m = 0_{\mathcal{L}(E,E)}$.

- **3.1.** Justifier qu'il existe $x \in E$ tel que $f^{m-1}(x) \neq 0_E$. Un vecteur x vérifiant cette propriété est fixé dans la suite.
- **3.2.** Montrer que la famille $(x, f(x), \ldots, f^{m-1}(x))$ est libre dans E.
- **3.3.** En déduire un minorant de $\dim E$.

➥ **SOLUTION de l'Exercice 2.**

□ **1**—

■ **1.1.** On vérifie facilement que l'application est linéaire. En effet, si $(x_1, \ldots, x_n)$ et $(y_1, \ldots, y_n)$ sont deux éléments de $\mathbf{K}^n$ et λ, μ deux éléments de $\mathbf{K}$, on a alors :

$$\begin{aligned} f(\lambda(x_1, \ldots, x_n) + \mu(y_1, \ldots, y_n)) &= f(\lambda x_1 + \mu y_1, \ldots, \lambda x_n + \mu y_n) \\ &= (0, \lambda x_1 + \mu y_1, \ldots, \lambda x_{n-1} + \mu y_{n-1}) \\ &= \lambda f((x_1, \ldots, x_n)) + \mu f((y_1, \ldots, y_n)). \end{aligned}$$

Donc $\boxed{f \text{ est linéaire.}}$ Un n-uplets est dans le noyau si $x_1 = x_2 = \ldots = x_{n-1} = 0$, donc le noyau est $\boxed{\text{Vect}(1, 0, \ldots, 0).}$ Enfin, l'image est composée des vecteurs dont la première coordonnée est nulle, donc $\boxed{\text{Im } f = \text{Vect } e_2, e_3, \ldots, e_n,}$ où $e_2, \ldots, e_n$ sont les $n-1$ derniers vecteurs de la base canonique de $\mathbf{K}^n$.

■ **1.2.** On remarque immédiatement que l'on fait apparaître un zéro à gauche en appliquant une fois f, donc n zéros en appliquant n fois f, en résumé : $f^n = 0$ donc $\boxed{f \text{ est nilpotent.}}$

□ **2**—

■ **2.1.** Voir TD sur les applications linéaires : nous avons prouvé que $\boxed{\deg \Delta(P) = \deg P - 1}$ pour tout élément $P \in \mathbf{K}[X]$.

■ **2.2.** Voir TD : nous avions prouvé que Δ est linéaire et clairement si $P \in \mathbf{K}_n[X]$ alors $\Delta(P) \in \mathbf{K}_n[X]$ puisque $\deg \Delta(P) < \deg P$.

■ **2.3.** Prenons $P \in \text{Ker } \Delta_n$. Alors pour tout $x \in \mathbf{R}$, $P(x) = P(x+1)$. En particulier $P(0) = P(1) = \ldots = P(n)$ pour tout $n \in \mathbf{N}$, notons $\alpha = P(0) = P(1) = \ldots = P(n)$ leur valeur commune. Alors $(P - \alpha)(n) = 0$ pour tout n donc $P - \alpha$ possède une infinité de racines. Ainsi, P est constant (égal à $P(0) = \alpha$) et $\boxed{\text{Ker } \Delta_n = \text{Vect}(1).}$

Le noyau est de dimension 1, l'image est donc de dimension $n + 1 - 1 = n$ d'après la formule du rang. Or, $\text{Im } \Delta_n \subset \mathbf{K}_{n-1}[X]$ d'après ce qui précède, donc par égalité des dimensions on conclut : $\boxed{\text{Im } \Delta_n = \mathbf{K}_{n-1}[X].}$

■ **2.4.** Puisque le degré diminue de un en appliquant Δ à un polynôme $P \in \mathbf{K}_n[X]$, on a $\Delta_n^n(P)$ constant disons égal à C. Or, $\Delta(C) = C - C = 0$ donc $\Delta_n^{n+1}(P) = 0$ et $\boxed{\Delta_n \text{ est donc nilpotent.}}$

□ **3**—

■ **3.1.** L'hypothèse contraire de « il existe $x \in E$ tel que $f^{m-1}(x) \neq 0_E$ » est « pour tout $x \in E$, $f^{m-1}(x) = 0_E$ » ce qui impliquerait $f^m = 0_{\mathcal{L}(E)}$ — contradiction. Donc $\boxed{\text{il existe } x \in E \text{ tel que } f^{m-1}(x) \neq 0_E}$.

■ **3.2.** Soit $(\lambda_0, \ldots, \lambda_{m-1}) \in \mathbf{K}^m$ tel que : $\lambda_0 x + \lambda_1 f(x) + \ldots + \lambda_m f^m(x) = 0$.

Alors appliquons f^{m-1} à cette identité. Nous obtenons : $\lambda_0 f^{m-1}(x) + 0 + \ldots + 0 = 0$ en utilisant que $f^m = 0$ et que pour tout $k \geqslant m$, on a aussi $f^k = 0$. Puisque $f^{m-1}(x) \neq 0$, une propriété du cours nous permet d'affirmer que $\lambda_0 = 0$. Ensuite, on recommence, on applique cette fois f^{m-2} dans l'égalité initiale qui nous livre $\lambda_1 f^{m-1}(x) + 0 = 0$ *etc.* ...

Ainsi, $\boxed{\text{la famille } (x, f(x), \ldots, f^{m-1}(x)) \text{ est libre.}}$

■ **3.3.** Nous avons donc une famille libre à m éléments, donc $\boxed{\dim E \geqslant m}$.

■

Exercice 3 **Informatique**

Recherche de zéro d'une fonction

□ **1**— Soient a et b deux réels, $f : [a, b] \to \mathbf{R}$ une fonction continue telle que : $f(a)f(b) < 0$.

■ **1.1.** Justifier que f s'annule sur $[a, b]$.

■ **1.2.** Écrire une fonction Python `rech_dicho` prenant en arguments une fonction `f`, deux flottants `a` et `b` tels que `f(a)f(b)<0` et une précision `eps` et qui renvoie un couple de réels encadrant un zéro de `f` à une précision `eps` près.

□ **2**— Soit f une fonction continue de $[0, 1]$ dans $[0, 1]$.

■ **2.1.** Montrer que f admet un point fixe (c'est-à-dire un réel c de $[0, 1]$ tel que $f(c) = c$).

■ **2.2.** Écrire une fonction Python `rech_pt_fixe` qui prend en argument une fonction `f` que l'on suppose continue de $[0, 1]$ dans $[0, 1]$, une précision `eps` et qui renvoie un couple de réels encadrant un point fixe de `f` à une précision `eps` près. *On pourra utiliser la fonction `rechdicho`.*

Recherche dans une liste

□ **3**— On propose l'algorithme suivant :

```
1  def rech_dicho(L,g,d,x):
2      if x>L[d]:
3          return d+1
4      else:
5          a=g
6          b=d
7          while a!=b:
8              c=(a+b)//2
9              if x<L[c]:
10                 b=c
11             else:
12                 a=c+1
13     return(a)
```

■ **3.1.** On prend `L = [2,4,5,7,7,9,10]`. Que renvoient les instructions suivantes ?

```
rech_dicho(L,1,5,6)
rech_dicho(L,0,5,1)
```

On donnera les valeurs prises par les variables `a` et `b` à chaque passage ligne 8.

■ **3.2.** Détailler clairement ce que fait le programme `rech_dicho`.

■ **3.3. [Bonus]—** Déterminer, en le justifiant, la complexité du programme, mesurée en nombre de comparaisons. On utilisera, si besoin est, la notation O, et on pourra exprimer cette complexité en fonction d'un ou plusieurs paramètres parmi `len(L)`, `g`, `d`, `x`.

□ **4**— Proposer un algorithme `tri_dicho` de tri par insertion **utilisant la fonction** `rech_dicho` pour trouver la position à laquelle insérer l'élément.

□ **5**— **[Bonus]—** Estimer le nombre **d'affectations** de `tri_dicho` ainsi que le nombre de **comparaisons** effectuées par l'algorithme `tri_dicho`. Comparer avec le tri par insertion classique en $O\left(n^2\right)$ dans le pire cas.

SOLUTION

Recherche de zéro d'une fonction

□ **1**—

■ **1.1.** La fonction f est continue et change de signe sur l'intervalle $[a, b]$ donc, d'après le théorème des valeurs intermédiaires,

f s'annule sur $[a, b]$.

■ **1.2.** La recherche par dichotomie d'un zéro d'une fonction revient à construire les suites (a_n) et (b_n) suivantes, définies par $a_0 = a$, $b_0 = b$ et pour tout n dans $\mathbf{N}$,

$$a_{n+1} = \begin{cases} \dfrac{a_n + b_n}{2} & \text{si } f(a_n)f\left(\dfrac{a_n + b_n}{2}\right) \leq 0, \\ a_n & \text{sinon} \end{cases}, \quad b_{n+1} = \begin{cases} b_n & \text{si } f(a_n)f\left(\dfrac{a_n + b_n}{2}\right) \leq 0, \\ \dfrac{a_n + b_n}{2} & \text{sinon} \end{cases}.$$

D'où le programme suivant :

```
def rech_pt_fixe(f,a,b,eps):
    """f est une fonction continue telle que f(a)*f(b)<0,
    avec a<b, eps un flottant >0"""
    an = a #Initialisation
    bn = b
    while bn-an>eps:
```

```
        cn=(an+bn)/2
        if f(an)*f(cn)<=0:
            bn = cn
        else:
            an=cn
    return an,bn
```

□ **2**—

■ **2.1.** Posons $g : x \mapsto f(x) - x$. Alors $g(0) = f(0) - 0 \geq 0$, $g(1) = f(1) - 1 \leq 1$. $0 \in [g(1), g(0)]$, g est continue sur l'intervalle $[0,1]$ donc, d'après le théorème des valeurs intermédiaires, on dispose de $\boxed{c \in [0,1] \text{ tel que } g(c) = 0, \textit{ i.e. } f(c) = c}$.

■ **2.2.** On va utiliser l'idée de la preuve de la question précédente, et appliquer la dichotomie à la fonction $x \mapsto f(x) - x$.

```
def rech_pt_fixe(f,x,eps):
    """f est une fonction de [0,1] dans [0,1], continue"""
    def g(x): #Definition de la fonction auxilliaire
        return f(x) - x
    return rech_dicho(g,0,1,eps)
```

Recherche dans une liste

□ **3**—

■ **3.1.**

```
L = [2,4,5,7,7,8,10]
rech_dicho(L,1,5,6)
rech_dicho(L,0,5,1)
```

renvoient 3 et 0.

■ **3.2.** Le programme `rech_dicho(L,g,d,x)` **recherche** et **renvoie** une **position** `i` entre `g` et `d` telle que `L[i-1]`$\leq$ `x` $\leq$ `L[i]`, sachant que l'on suppose `L[g:d+1]` triée.

□ **4**— Si a_k et b_k sont les valeurs de `a` et `b` à la k-ième étape, alors $\frac{a_k + b_k}{2} - 1 \leq c_k \leq \frac{a_k + b_k}{2}$. Alors selon les deux cas de figure du test `if`, on obtient

$$b_{k+1} - a_{k+1} = c_k - a_k \leq \frac{a_k + b_k}{2} - a_k \leq \frac{b_k - a_k}{2},$$

$$\text{ou bien } b_{k+1} - a_{k+1} = b_k - (c_k + 1) = b_k - c_k - 1 \leq b_k - \frac{a_k + b_k}{2} \leq \frac{b_k - a_k}{2},$$

donc, par une récurrence immédiate, $b_k - a_k \leq 2^{-k}(d - g + 1)$, donc le programme s'arrête lorsque $b_k - a_k < 1$, en particulier lorsque $2^{-k}(d - g + 1) < 1$, *i.e.* $k \geq \log_2(d - g + 1)$, d'où une complexité en $\boxed{O(\ln(d - g)).}$

□ **5**— Pour trier par insertion une liste `L`, on part du premier élément de `L`, puis on **insère** le deuxième élément avant ou après ce premier élément, puis, de manière générale, lorsqu'on a une liste dont les k premiers éléments sont triés, on insère le $k+1$-ème dans ces k premiers éléments. Cette insertion se fait en deux étapes : la recherche de la position de l'élément à proprement parler, puis l'insertion de l'élément. On va effectuer cette insertion avec des permutations successives. On propose donc l'algorithme suivant :

```
1 def tri_dicho(L):
2     for k in range(1,len(L)):
3         if L[k]<L[k-1]:
4             i = rech_dicho(L,0,k-1,L[k])
5             for j in range(k,i,-1):
6                 L[j],L[j-1] = L[j-1],L[j]
```

□ **6**— Dans le programme précédent et si $n =$ `len(L)`,

✓ Il y a une première boucle à $n - 1$ étapes.

✓ Dans la boucle, on appelle `rech_dicho`, qui effectue $O(\ln(n))$ comparaisons, d'où $O(n \ln(n))$ comparaisons.

✓ Dans la boucle, on effectue $k - i = O(k)$ affectations, d'où $O(\sum_{k=1}^{n} k) = O(n^2)$ affectations.

Le tri par insertion classique est similaire, sauf pour la recherche de la position, pour laquelle on effectue $O(k)$ comparaisons. D'où, pour le tri par insertion classique, $O(n^2)$ affectations **et** $O(n^2)$ comparaisons. Ainsi, le tri par insertion dichotomique permet un gain de complexité en termes de comparaisons, mais pas en termes d'affectations.