DEVOIR SURVEILLÉ # 11

le Samedi 25/09/2021, 8h ⟶ 10h30

Consignes 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.

L'usage de la calculatrice est interdit.

Exercice 1 (Solution : 3) Soit $n \in \mathbf{N}$ et on note

$$P = X^n - 1, \quad Q = 1 + X + \cdots + X^{n-1}.$$

1. Déterminer les racines complexes de P. Quelle est leur multiplicité ?
2. En déduire la décomposition de P sous forme de produit de polynômes irréductibles.
3. Déterminer une relation simple entre les polynômes P et Q, puis en déduire la décomposition en produit de polynômes irréductibles de Q.
4. En calculant Q(1), déterminer

$$\prod_{k=1}^{n-1} \sin\left(\frac{k\pi}{n}\right).$$

Problème 1 ***Quelques endomorphismes nilpotents.*** (Solution : 3) Dans cet exercice la notation **K** désignera **R** ou **C**. Si E est un espace vectoriel et $f \in \mathscr{L}(E)$, 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 $f^m = 0_{\mathscr{L}(E)}$.
- Pour $n \geqslant 1$, une matrice $A \in \mathfrak{M}_{n,n}(\mathbf{K})$ est dite *nilpotente* s'il existe un entier naturel m tel que $A^m = 0_{\mathfrak{M}_{n,n}(\mathbf{K})}$.

1. **(«Right–shift»)** Soit $n \geqslant 1$. 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 est un endomorphisme de $\mathbf{K}^n$). Déterminer son noyau et son image.
   - **1.2)** >_ On représente un élément $(x_1, x_2, \ldots, x_{n-1}, x_n) \in \mathbf{K}^n$ par une liste `L = [x1, x2, ..., xn]`.
     - **i)** Écrire une fonction d'en-tête `calc_f(L)` prenant en argument une liste `L`, et renvoyant la liste correspondant à `f(L)`.
     - **ii)** En déduire une fonction d'en-tête `nilpo_calc_f(L)` qui calcule pour $x = (x_1, \ldots, x_n) \in \mathbf{K}^n$ fixé correspondant à `L` le plus petit entier m tel que $f^m(x) = 0_{\mathbf{K}^n}$.
   - **1.3)** Montrer que f est nilpotent.
2. **(Endomorphisme associé à une matrice nilpotente)** Dans cette question, on considère

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix},$$

et on définit

$$f \left| \begin{array}{ccc} \mathfrak{M}_{3,1}(\mathbf{K}) & \longrightarrow & \mathfrak{M}_{3,1}(\mathbf{K}), \\ X & \longmapsto & AX. \end{array} \right.$$

2.1) Monter que la matrice A est nilpotente.
2.2) Établir que : $\forall k \in \mathbf{N}, \quad f^k(X) = A^k X$.
2.3) Que dire sur f ?

3. (Différences finies) On définit $\Delta : \left| \begin{array}{ccc} \mathbf{K}[X] & \longrightarrow & \mathbf{K}[X], \\ P & \longmapsto & P(X+1) - P(X). \end{array} \right.$

3.1) Calculer $\Delta(1), \Delta(X), \Delta(X^2)$. Conjecturer une relation entre le degré de P et celui de $\Delta(P)$.
3.2) Montrer cette relation.
3.3) Dans la suite, on considère $n \geqslant 1$. Montrer que $\Delta\big|_{\mathbf{K}_n[X]} : P \in \mathbf{K}_n[X] \longmapsto \Delta(P)$ est un endomorphisme de $\mathbf{K}_n[X]$. Nous le noterons Δ_n dans la suite.
3.4) Montrer que Δ_n est nilpotent.
3.5) Si $P \in \operatorname{Ker}\Delta_n$, montrer que $Q = P - P(0)$ possède une infinité de racines. En déduire que $\operatorname{Ker}\Delta_n$ est l'ensemble des polynômes constants.
3.6) En déduire le rang, puis l'image de Δ_n.

4. (Famille cyclique associée à un endomorphisme nilpotent) Soient E un $\mathbf{K}$-espace vectoriel de dimension finie, $f \in \mathscr{L}(E,E)$ nilpotent, et m le plus petit entier strictement positif tel que : $f^m = 0_{\mathscr{L}(E,E)}$.
4.1) 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.
4.2) Montrer que la famille $\mathscr{L} = (x, f(x), \dots, f^{m-1}(x))$ est libre dans E.
4.3) En déduire un minorant de $\dim E$.

Exercice 2 ***Type G2E. ADN et manipulations à l'aide de chaînes*** (Solution : 5) Un *codon* est constitué de 3 nucléotides, et a pour base azotée soit A, soit T, soit G, soit C. En Python, on le codera donc par une chaîne de caractères de longueur 3, composée uniquement des lettres `A`, `T`, `G`, `C`. Un *codon-stop* est un codon de la forme `TAG`, `TAA` ou `TGA`.

1. Écrire une fonction d'en-tête `verif_codon(c)` qui prend en entrée une chaîne `c`, et qui renvoie `True` si c'est bien un codon et `False` sinon.
2. Écrire une fonction d'en-tête `cherche_codon_stop(c)` qui prend en entrée une chaîne `c`, et qui renvoie la position du premier codon stop présent dans `c`, si `c` en possède un, et `-1` si elle n'en possède pas.
3. (Complémentarité) Les nucléotides sont *complémentaires*, au sens où `T` (*resp.* `C`) est le complémentaire de `A` (*resp.* `G`) (et inversement).
3.1) Écrire une fonction d'en-tête `complement(c)` qui prend en argument un nucléotide (donc une chaîne de longueur 1 égale à `'A'`, `'T'`, `'G'` ou `'C'`) et qui retourne le complément de `c`.
3.2) Écrire une fonction d'en-tête `est_complement_brun(c1, c2)` qui prend en argument deux brins `c1`, `c2` de même taille et qui vérifie s'ils sont complémentaires. Si oui, elle renvoie `True`, sinon elle renvoie `False`.

4. (Croisement) On rappelle sur un exemple le phénomène de croisement entre deux brins au niveau d'une position admissible k sur le brin. Par exemple si les brins : `'ATTGTC'` et `'CGCTGA'` se croisent à la position 3, les deux nouveaux brins sont : `'ATTTGA'` et `'CGCGTC'`, il y a donc échange des deux nucléotides à la position `k`. On rappelle que l'on ne peut changer un élément d'une chaîne de caractère.
Écrire une fonction d'en-tête `croisement(c1, c2, k)` qui prend en paramètre deux brins et une position k et qui fait le croisement, *i.e.* la fonction renverra deux chaînes de caractère correspondantes aux deux nouveaux bruns.

On rappelle qu'on ne peut modifier une chaîne de caractères, une fois définie. On pourra se référer aux commandes ci-après, et observer leur résultat

```
>>> s = "superleDS"
>>> L = list(s)
>>> L.append("1")
>>> L
['s', 'u', 'p', 'e', 'r', 'l', 'e', 'D', 'S', '1']
>>> "".join(L)
'superleDS1'
```

CORRECTION

Solution (exercice 1) (Énoncé : 1)

1. Voir le cours, en utilisant la forme exponentielle de z. On trouve les racines de l'unité ci-après :

$$\boxed{\left\{e^{\frac{2ik\pi}{n}},\ k\in[\![0,n-1]\!]\right\}}.$$

Les racines sont simples, puisque distinctes et au nombre de n (ou bien constater que $P'=nX^{n-1}$ et qu'aucune des racines trouvées n'annulent le dérivée $P'\left(e^{\frac{2ik\pi}{n}}\right)=ne^{\frac{2ik(n-1)\pi}{n}}\neq 0$ pour tout k).

2. On déduit immédiatement, puisque P est de coefficient dominant égal à 1, que

$$\boxed{X^n-1=\prod_{k=0}^{n-1}\left(X-e^{\frac{2ik\pi}{n}}\right)}.$$

3. Pour $x\neq 1$, nous avons

$$\frac{x^n-1}{x-1}=Q(x),$$

d'après la formule de sommation de termes géométriques. Donc

$$x^n-1=(x-1)Q(x),$$

cette formule est même vraie si $x=1$. On obtient alors

$$\boxed{P=(X-1)Q}.$$

Ou de manière équivalente

$$(X-1)\left(\prod_{k=1}^{n-1}\left(X-e^{\frac{2ik\pi}{n}}\right)-Q\right)=0.$$

Comme $X-1\neq 0$, nous déduisons la factorisation suivante :

$$\boxed{Q=\prod_{k=1}^{n-1}\left(X-e^{\frac{2ik\pi}{n}}\right)}.$$

4. Évaluons l'égalité précédente en 1, on obtient alors

$$Q(1)=\prod_{k=1}^{n-1}\left(1-e^{\frac{2ik\pi}{n}}\right),$$

en mettant l'angle moitié en facteur, nous obtenons :

$$Q(1)=n=\prod_{k=1}^{n-1}e^{\frac{ik\pi}{n}}\left(-2i\sin\left(\frac{k\pi}{n}\right)\right),$$

d'où

$$n=(-2i)^{n-1}\prod_{k=1}^{n-1}e^{\frac{ik\pi}{n}}\times\prod_{k=1}^{n-1}\sin\left(\frac{k\pi}{n}\right),$$

or,

$$\prod_{k=1}^{n-1}e^{\frac{ik\pi}{n}}=e^{\frac{i\pi}{n}\sum_{k=1}^{n-1}k}=e^{\frac{(n-1)i\pi}{2}}=i^{n-1},$$

puisque $e^{\frac{i\pi}{2}}=i$. Donc

$$n=(-2i)^{n-1}i^{n-1}\prod_{k=1}^{n-1}\sin\left(\frac{k\pi}{n}\right),$$

et $(-2i)^{n-1}i^{n-1}=2^n(-1)^n(i^2)^{n-1}=2^n(-1)^n(-1)^{n-1}=-2^n$, et l'on déduit finalement

$$\boxed{\prod_{k=1}^{n-1}\sin\left(\frac{k\pi}{n}\right)=\boxed{-\frac{n}{2^n}}}.$$

Solution (problème 1) (Énoncé : 1)

1. 1.1) On vérifie facilement que l'application est linéaire. En effet, si $(x_1,\dots,x_n)$ et $(y_1,\dots,y_n)$ sont deux éléments de $\mathbf{K}^n$ et λ,μ deux éléments de $\mathbf{K}$, on a alors :

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

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) i) On peut par exemple supprimer le dernier élément puis insérer 0 en position 0 à l'aide d'`insert`. On peut aussi :

```python
def calc_f(L):
    M = L[:] # copie profonde de L dans M
    del M[len(L)-1]
    return [0]+M
```

ii)
```python
def nilp_calc_f(L):
    n = len(L)
    m = 0
    while L != [0]*n:
        # si L n'est pas la liste nulle, on
        ↪ recommence
        L = calc_f(L)
        m += 1
    return m
```

1.3) 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) Un simple calcul montre que $A^2 = \begin{pmatrix} 0 & 0 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, puis que $\boxed{A^3 = 0}$. La matrice A est donc nilpotente.

2.2) Faisons une récurrence sur k.

Initialisation. Pour $k = 0$: $A^0X = I_3X = X$ et $f^0(X) = X$ pour tout $X \in \mathfrak{M}_{3,1}(\mathbf{K})$, d'où le résultat.

Hérédité. Supposons la propriété vraie au rang $k \in \mathbf{N}$, alors $f^{k+1}(X) = f \circ f^k(X) = f(A^kX) = AA^kX = A^{k+1}X$ pour tout $X \in \mathfrak{M}_{3,1}(\mathbf{K})$. D'où par principe de récurrence : $\boxed{\forall k \in \mathbf{N}, \quad f^k(X) = A^kX.}$

2.3) Donc en combinant les deux questions f^3 est l'endomorphisme nul, donc $\boxed{f \text{ est nilpotent.}}$

3. 3.1) On a $\Delta(1) = 1-1 = 0$, $\Delta(X) = X+1-X = 1$ et $\Delta(X^2) = (X+1)^2 - X^2 = 2X+1$. On conjecture $\deg\Delta(P) = \deg P - 1$ pour tout P non constant, et $\deg\Delta(P) = -\infty$ si P est constant.

3.2)
- Si $P = C \in \mathbf{K}$ est constant ou nul, on a $\Delta(P) = C - C = 0$ donc $\Delta(P)$ est bien de degré $-\infty$.
- Sinon, notons $P = \sum_{k=0}^{p} a_k X^k$ si $p \geqslant 1$ est le degré de P. Alors

$$\Delta(P) = \sum_{k=0}^{p} a_k (X+1)^k - \sum_{k=0}^{p} a_k X^k = \sum_{k=0}^{p} a_k \left[(X+1)^k - X^k\right].$$

Le coefficient dominant de $\Delta(P)$ est donc celui de

$$a_p \left[(X+1)^p - X^p\right].$$

Mais d'après la formule du binôme

$$a_p \left[(X+1)^p - X^p\right] = a_p \left[\sum_{k=0}^{p} \binom{p}{k} X^k - X^p\right].$$

C'est donc, en simplifiant

$$a_p \sum_{k=0}^{p-1} \binom{p}{k} X^k.$$

Comme $a_p \binom{p}{p-1} \neq 0$, ce polynôme est bien de degré $p-1$, donc $\sum_{k=0}^{p} a_k \left[(X+1)^k - X^k\right]$ aussi et $\boxed{\deg\Delta(P) = \deg P - 1.}$

3.3) Nous avons $\Delta_n(\mathbf{K}_n[X]) \subset \mathbf{K}_n[X]$ par des considérations de degré. Reste à montrer la linéarité. Soient donc $P, Q \in \mathbf{K}_n[X]$ et $\lambda, \mu \in \mathbf{K}$. Alors

$$\begin{aligned}\Delta_n(\lambda P + \mu Q) &= (\lambda P + \mu Q)(X+1) - (\lambda P + \mu Q)(X) \\ &= \lambda(P(X+1) - P(X)) + \mu(Q(X+1) - Q(X)) \\ &= \lambda\Delta_n(P) + \mu\Delta_n(Q).\end{aligned}$$

Donc $\boxed{\Delta_n \text{ est un endomorphisme de } \mathbf{K}_n[X].}$

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

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

3.6) Le noyau est de dimension 1, l'image est donc de dimension $n+1-1 = n$ d'après la formule du rang. Or, $\operatorname{Im}\Delta_n \subset \mathbf{K}_{n-1}[X]$ d'après le résultat sur le degré, et $\dim \mathbf{K}_{n-1}[X] = n$, donc par égalité des dimensions on conclut : $\boxed{\operatorname{Im}\Delta_n = \mathbf{K}_{n-1}[X].}$

4. 4.1) 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 $f^m = 0_{\mathscr{L}(E)}$ — contradiction. Donc $\boxed{\text{il existe } x \in E \text{ tel que } f^{m-1}(x) \neq 0_E}$.

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

4.3) Nous avons donc une famille libre à m éléments, donc $\boxed{\dim E \leqslant m}$.

Solution (exercice 2) (Énoncé : 2)

1. On peut par exemple créer une variable globale LETTRES qui contient les lettres possibles pour les codons.

```python
LETTRES = ["A", "T", "C", "G"]
def verif_codon(c):
    """
    retourne True si c est un codon
    """
    verif_lettres = True
    for x in c:
        if x not in LETTRES:
            verif_lettres = False
    return len(c) == 3 and verif_lettres

c1 = 'ABCD'
c2 = 'TAG'
```

Pour c1 la fonction renvoie False, pour c2 la fonction renvoie True.

2. C'est aun algorithme du type recherche de mot dans une chaîne.

```python
CODONS_STOP = ["TAA", "TAG", "TGA"]
def cherche_codon_stop(c):
    """
    retourne l'indice d'apparition du codon dans c, -1 s'il n'est pas présent
    """
    if len(c) < 3:
        return -1
    else:
        for i in range(len(c)-3+1):
            if c[i:i+3] in CODONS_STOP:
                return i
        # si on est rentré dans aucun return, c'est qu'on ne l'a pas trouvé
        return -1
```

Pour `c1` la fonction renvoie -1, pour `c2` la fonction renvoie bien 0.

3. (Complémentarité) Les nucléotides sont *complémentaires*, au sens où `T` (*resp.* `C`) est le complémentaire de `A` (*resp.* `G`) (et inversement).

3.1) On aligne plusieurs tests logiques.

```python
def complement(c):
    """
    retourne le complément de la lettre c
    """
    if c == "A":
        return "T"
    elif c == "T":
        return "A"
    elif c == "C":
        return "G"
    else:
        return "C"
```

3.2) On parcourt par exemple le brun `c1` et on regarde si chaque lettre est bien le complémentaire de l'autre.

```python
def est_complement_brun(c1, c2):
    """
    vérifie la complémentarité lettre par lettre
    """
    for i in range(len(c1)):
        if complement(c1[i]) != c2[i]:
            return False
    return True

c3 = "ATGC"
c4 = "TACG"
```

Par exemple, la fonction retourne True pour `c3, c4`.

4. (Croisement)

```python
def croisement(c1, c2, k):
    """
    fait le croisement en la position k entre les bruns c1, c2
    renvoie deux nouvelles chaînes sans modifier c1, c2
    """
    L_1 = []
    L_2 = []
    for i in range(k-1):
        # début de chaîne
        L_1.append(c1[i])
        L_2.append(c2[i])
    # étape de croisement
    L_1.append(c2[k])
    L_2.append(c1[k])
    for i in range(k+1, len(c1)):
        # fin de chaîne
        L_1.append(c1[i])
        L_2.append(c2[i])
    # Il reste à concaténer les lettres des listes pour en faire une chaîne, à l'aide de join
    return "".join(L_1), "".join(L_2)
c3 = "ATGC"
c4 = "TACG"
```

Par exemple, la fonction retourne ('AGC', 'TCG') pour `c3, c4` en position `k = 1`.