# Correction
# DS 3 Mathématiques
# Sujet B

## Problème

Si $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ sont deux suites réelles, indexées à partir de 0, on définit une nouvelle suite $(w_n)_{n\in\mathbb{N}}$ en posant :

$$\forall n \in \mathbb{N}, \quad w_n = \sum_{k=0}^{n} u_k v_{n-k} \tag{1}$$

On appelle $(w_n)_{n\in\mathbb{N}}$ le **produit de convolution** de $(u_n)_{n\in\mathbb{N}}$ avec $(v_n)_{n\in\mathbb{N}}$. Si dans la formule ci-desus on prend $\forall n \in \mathbb{N}$, $v_n = u_n$, on parle simplement du produit de convolution de $(u_n)_{n\in\mathbb{N}}$ avec elle-même.

1. (a) Soit $n \in \mathbb{N}$. Calculer

$$\sum_{k=1}^{n} k(n-k) \tag{2}$$

*Correction.* On développe et on utilise les formules habituelles :

$$\begin{aligned}
\sum_{k=1}^{n} k(n-k) &= \sum_{k=1}^{n} (kn - k^2) \\
&= \sum_{k=1}^{n} kn - \sum_{k=1}^{n} k^2 \\
&= n\sum_{k=1}^{n} k - \sum_{k=1}^{n} k^2 \\
&= n \times \frac{n(n+1)}{2} - \frac{n(n+1)(2n+1)}{6} \\
&= \frac{n(n+1)}{6}\Big(3n - (2n+1)\Big) \\
&= \frac{n(n+1)}{6}(n-1) \\
&= \frac{n(n-1)n+1)}{6}
\end{aligned}$$

□

(b) On considère la suite $(e_n)_{n\in\mathbb{N}}$ donnée par $\forall n \in \mathbb{N}$, $e_n = n$. Calculer le produit de convolution de $(e_n)_{n\in\mathbb{N}}$ avec elle-même.

*Correction.* Par définition il s'agit de calculer le terme $w_n$ avec

$$w_n = \sum_{k=0}^{n} k(n-k)$$

Mais cette somme est la même que la précédente à la différence près qu'elle part de $k = 0$. Mais pour $k = 0$ le terme sous la somme est nul. Donc $w_n$ est bien égal à la somme trouvée précédemment. □

2. Soit un nombre $a \in \mathbb{R}$. On considère la suite $(\alpha_n)_{n\in\mathbb{N}}$ définie par $\forall n \in \mathbb{N}$, $\alpha_n = a^n$. Calculer le produit de convolution de $(\alpha_n)_{n\in\mathbb{N}}$ avec elle-même.

*Correction.* Il s'agit de calculer, pour $n \in \mathbb{N}$, le terme $w_n$ défini par

$$w_n = \sum_{k=0}^{n} a^k a^{n-k}$$

mais sous la somme cela se simplifie en $a^n$ qui ne dépend pas de $k$. Il y a alors $n+1$ termes égaux à $a^n$ donc on trouve

$$w_n = (n+1)a^n \tag{3}$$

□

3. Soit un autre nombre $b \in \mathbb{R}$, de même on considère la suite $(\beta_n)_{n\in\mathbb{N}}$ définie par $\forall n \in \mathbb{N}$, $\beta_n = b^n$. Montrer que le produit de convolution de $(\alpha_n)_{n\in\mathbb{N}}$ avec $(\beta_n)_{n\in\mathbb{N}}$ est la suite $(w_n)_{n\in\mathbb{N}}$ donnée par :

$$\forall n \in \mathbb{N}, \quad w_n = \begin{cases} (n+1)a^n & \text{si } a = b \\ \dfrac{a^{n+1} - b^{n+1}}{a-b} & \text{si } a \neq b \end{cases} \tag{4}$$

*Correction.* On écrit

$$w_n = \sum_{k=0}^{n} a^k b^{n-k}$$

Si $a = b$ cela se ramène en fait au cas prédédent où on trouvait bien $(n+1)a^n$. Sinon cela ressemble beaucoup à la formule écrite telle quelle dans le cours

$$a^n - b^n = (a-b)\sum_{k=0}^{n-1} a^{n-1-k}b^k$$

qui est en fait la même (quitte à l'appliquer à $(b,a)$ au lieu de $(a,b)$, ou à faire le changement d'indice $j = n-1-k$, même principe que dans le binôme de Newton) que

$$a^n - b^n = (a-b)\sum_{k=0}^{n-1} a^k b^{n-1-k}$$

Écrite avec $n+1$ à la place de $n$ c'est précisément

$$a^{n+1} - b^{n+1} = (a-b)\sum_{k=0}^{n} a^k b^{n-k}$$

et donc à droite on reconnait $(a-b)w_n$. Si $a \neq b$ alors on peut en plus diviser par $a-b$ et on trouve bien

$$w_n = \frac{a^{n+1} - b^{n+1}}{a-b}$$

□

4. On définit la suite $(\varepsilon_n)_{n\in\mathbb{N}}$ par

$$\forall n \in \mathbb{N}, \quad \varepsilon_n = \begin{cases} 1 & \text{si } n = 0 \\ 0 & \text{sinon} \end{cases} \tag{5}$$

Montrer que pour toute suite $(u_n)_{n\in\mathbb{N}}$, le produit de convolution de $(u_n)_{n\in\mathbb{N}}$ avec $(\varepsilon_n)_{n\in\mathbb{N}}$ est la suite $(w_n)_{n\in\mathbb{N}}$ avec $\forall n \in \mathbb{N}$, $w_n = u_n$.

*Correction.* On écrit d'abord :

$$w_n = \sum_{k=0}^{n} u_k \varepsilon_{n-k}$$

Mais sous cette somme beaucoup de termes sont nuls ! En effet les seuls termes non nuls sont pour $n-k = 0$ (auquel cas $\varepsilon_{n-k} = 1$ c'est à dire en fait $k = n$. La somme n'a donc qu'un seul terme :

$$w_n = u_n \times 1 = u_n$$

et cela pour tout $n \in \mathbb{N}$. □

5. Pour deux nombres $(c,d) \in \mathbb{R}^2$, on définit les suites $(\gamma_n)_{n\in\mathbb{N}}$ et $(\delta_n)_{n\in\mathbb{N}}$ par :

$$\forall n \in \mathbb{N}, \quad \gamma_n = \frac{c^n}{n\,!} \quad \delta_n = \frac{d^n}{n\,!} \tag{6}$$

Montrer que, pour leur produit de convolution $(w_n)_{n\in\mathbb{N}}$,

$$\forall n \in \mathbb{N}, \quad w_n = \frac{(c+d)^n}{n\,!} \tag{7}$$

*Correction.* Soit $n \in \mathbb{N}$, écrivons $w_n$ :

$$w_n = \sum_{k=0}^{n} \frac{c^k}{k\,!} \times \frac{d^{n-k}}{(n-k)\,!}$$

et cela ressemble au binôme de Newton ; mais d'autre part

$$\frac{(c+d)^n}{n\,!} = \frac{1}{n\,!} \sum_{k=0}^{n} \binom{n}{k} c^k d^{n-k}$$

Remplaçant les coefficients binomiaux par leur valeur, si $0 \leqslant k \leqslant n$ :

$$\binom{n}{k} = \frac{n\,!}{k\,!(n-k)\,!}$$

on retrouve

$$\frac{(c+d)^n}{n\,!} = \frac{1}{n\,!} \sum_{k=0}^{n} \frac{n\,!}{k\,!(n-k)\,!} c^k d^{n-k} = \sum_{k=0}^{n} \frac{1}{k\,!(n-k)\,!} c^k d^{n-k} = w_n$$

□

6. **[Informatique]** On souhaite écrire un programme Python permettant de vérifier une partie des calculs précédents. On note plutôt avec $i \in \mathbb{N}$ l'*indice* des suites (ce n'est qu'une variable muette) et pour un nombre $n$ donné, on représente une suite $(u_i)_{i\in\mathbb{N}}$ par une liste `L` contenant les $n+1$ premiers termes de la suite, c'est à dire les termes $u_0$, $u_1$, …, $u_n$. Ainsi `L[i]` est exactement le terme $u_i$ et cela pour tout $0 \leqslant i \leqslant n$. Par exemple si $n = 5$ alors la suite $(e_i)_{i\in\mathbb{N}}$ est représentée par la liste `[0, 1, 2, 3, 4, 5]` et la suite $(\delta_i)_{i\in\mathbb{N}}$ est représentée par `[1, 0, 0, 0, 0, 0]`.

(a) Écrire une fonction `epsilon(n)` qui prend en argument un nombre `n` et qui renvoie la liste des $n+1$ premiers termes de la suite $(\varepsilon_i)_{i\in\mathbb{N}}$ de la question 4.

*Correction.* Plusieurs possibilités ; sur le modèle des questions suivantes (en initialisant une liste remplie de zéros) on peut faire tout simplement

```
def epsilon(n):
    L = [0] * (n+1)
    L[0] = 1
    return L
```

sinon on se retrouve à faire des programmes qui traduisent aussi bien la définition mathématique mais sont plus longs, par exemple

```
def epsilon(n):
    L = []
    for i in range(n+1):
        if i == 0:
            L.append(1)
        else:
            L.append(0)
    return L
```

□

(b) Écrire une fonction `e(n)` qui prend en argument un nombre `n` et qui renvoie la liste des $n+1$ premiers termes de la suite $(e_i)_{i\in\mathbb{N}}$ de la question 1b.

*Correction.* Idem plusieurs possibilités pour remplir une liste `L` avec `L[i] = i`. Sur ce même modèle :

```
def e(n):
    L = [0] * (n+1)
    for i in range(n+1):
        L[i] = i
    return L
```

ou bien une boucle avec `append`

```
def e(n):
    L = []
    for i in range(n+1):
        L.append(i)
    return L
```

ou bien directement en compréhension (type TP 5 exercice 3)

```
def e(n):
    return [i for i in range(n+1)]
```

□

(c) On considère la fonction suivante, prenant deux arguments (des nombres) `c` et `n` :

```
def manquedeserieux(c, n):
    L = [0] * n
    L[0] = c
    for i in range(1, n):
        L[i] = L[i-1] * c / (i+1)
    return L
```

i. Si on suppose que la variable `c` est un nombre, que renvoie `manquedeserieux(c, 2)` ? Et `manquedeserieux(c, 3)` ?

*Correction.* D'abord pour `n = 2` : au départ `L` est la liste `[0, 0]` puis à la ligne suivante `[c, 0]`. La boucle `for` est exécutée une seule fois, avec `i = 1`, qui remplit `L[1]` à `c*c/2`. Donc la fonction renvoie la liste `[c, c*c/2]`.
Puis pour `n = 3` : de même avant la boucle `L` est `[c, 0, 0]` puis au passage dans la boucle avec `i = 1` cela devient `[c, c*c/2, 0]` et enfin avec `i = 2` la liste renvoyée est `[c, c*c/2, c*c*c/6]`. □

ii. Corriger la fonction pour écrire une fonction `gamma(c, n)` qui renvoie la liste des $n+1$ premiers termes de la suite $(\gamma_n)_{n\in\mathbb{N}}$ de la question 5 avec le paramètre $c$.

*Correction.* La fonction précédente retourne bien une liste de termes du type $\frac{c^i}{i!}$ mais les bornes ne sont pas bonnes (pour $i = 0$ ce terme doit être égal à 1) et elle retourne seulement $n$ termes mais on en veut $n+1$. Soyons sérieux ! On vérifie comme à la question précédente que la fonction corrigée est :

```
def gamma(c, n):
    L = [0] * (n+1)
    L[0] = 1
    for i in range(1, n+1):
        L[i] = L[i-1] * c / i
    return L
```

□

(d) Écrire une fonction `convole(L, M, n)`, qui prend en argument un entier `n` et deux listes `L`, `M` de taille (au moins) $n+1$ et qui renvoie le terme $w_n$ du produit de convolution $(w_n)_{n\in\mathbb{N}}$ de $(u_n)_{n\in\mathbb{N}}$ avec $(v_n)_{n\in\mathbb{N}}$.

*Correction.* Il s'agit simplement de calculer la somme des termes `L[i] * M[n-i]`. Si on demande que les listes contiennent $n+1$ termes c'est justement pour pouvoir faire varier `i` de 0 à $n$, bornes incluses, et que les indices des listes coïncident bien avec les indices des suites. C'est donc :

```
def convole(L, M, n):
    s = 0
    for i in range(n+1):
        s = s + L[i] * M[n-i]
    return s
```

(les parenthèses autour de la multiplication ne sont pas nécessaires, à cause de la priorité des opérations). □

(e) Utiliser la fonction précédente pour écrire une fonction `convolution(L, M, n)` qui prend en argument deux listes `L`, `M` représentant les $n+1$ premiers termes de suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ et qui renvoie une liste représentant les $n+1$ premiers termes de la suite $(w_n)_{n\in\mathbb{N}}$ qui est leur produit de convolution.

*Correction.* Standard : on veut remplir une liste disons `C` où `C[i]` est donné par `convole(L, M, i)` (ce n'est pas une suite définie par récurrence). On peut donc faire

```
def convolution(L, M, n):
    C = [0] * (n+1)
    for i in range(n+1):
        C[i] = convole(L, M, i)
    return C
```

ou bien avec `append`

```
def convolution(L, M, n):
    C = []
    for i in range(n+1):
        C.append(convole(L, M, i))
    return C
```

ou bien en compréhension

```
def convolution(L, M, n):
    return [convole(L, M, i) for i in range(n+1)]
```

□

7. On souhaite montrer que le produit de convolution est commutatif. Pour cela on fixe des suites $(u_n)_{n\in\mathbb{N}}$, $(v_n)_{n\in\mathbb{N}}$ et on note $(w_n)_{n\in\mathbb{N}}$ leur produit de convolution. De même on note $(w'_n)_{n\in\mathbb{N}}$ le produit de convolution de $(v_n)_{n\in\mathbb{N}}$ avec $(u_n)_{n\in\mathbb{N}}$. Montrer que $\forall n \in \mathbb{N}$, $w_n = w'_n$.

*Correction.* Il s'agit de comparer, pour tout $n \in \mathbb{N}$, d'une part

$$w_n = \sum_{k=0}^{n} u_k v_{n-k}$$

et d'autre part

$$w'_n = \sum_{k=0}^{n} v_k u_{n-k}$$

Mais ce sont les mêmes à un changement d'indice près ! Si dans l'expression de $w_n$ on pose $j = n - k$ alors (donc $k = n - j$) cela devient

$$w_n = \sum_{j=0}^{n} u_{n-j} v_j = \sum_{j=0}^{n} v_j u_{n-j}$$

qui est donc bien égal à $w'_n$, au renommage d'indice près. □

8. On souhaite montrer que le produit de convolution est associatif. Pour cela on fixe trois suites $(x_n)_{n\in\mathbb{N}}$, $(y_n)_{n\in\mathbb{N}}$, $(z_n)_{n\in\mathbb{N}}$. D'une part on forme le produit de convolution $(r_n)_{n\in\mathbb{N}}$ de $(x_n)_{n\in\mathbb{N}}$ avec $(y_n)_{n\in\mathbb{N}}$, puis $(w_n)_{n\in\mathbb{N}}$ le produit de convolution de $(r_n)_{n\in\mathbb{N}}$ avec $(z_n)_{n\in\mathbb{N}}$. D'autre part, on forme d'abord le produit de convolution $(s_n)_{n\in\mathbb{N}}$ de $(y_n)_{n\in\mathbb{N}}$ avec $(z_n)_{n_i n\mathbb{N}}$, puis on forme le produit de convolution $(w'_n)_{n\in\mathbb{N}}$ de $(x_n)_{n\in\mathbb{N}}$ avec $(s_n)_{n\in\mathbb{N}}$. Montrer que $\forall n \in \mathbb{N}$, $w_n = w'_n$.

*Correction.* Il s'agit de comparer, pour tout $n \in \mathbb{N}$, d'une part

$$w_n = \sum_{k=0}^{n} r_k z_{n-k} = \sum_{k=0}^{n} \left( \sum_{i=0}^{k} x_i y_{k-i} \right) z_{n-k}$$

et d'autre part

$$w'_n = \sum_{k=0}^{n} x_k s_{n-k} = \sum_{k=0}^{n} x_k \left( \sum_{i=0}^{n-k} y_i z_{n-k-i} \right)$$

Or on a d'abord

$$w_n = \sum_{k=0}^{n} \left( \sum_{i=0}^{k} x_i y_{k-i} z_{n-k} \right)$$

puis on intervertit la doubles somme ce qui donne (c'est la somme sur tous les $0 \leqslant i, k \leqslant n$ avec en plus $i \leqslant k$)

$$w_n = \sum_{i=0}^{n} \left( \sum_{k=i}^{n} x_i y_{k-i} z_{n-k} \right)$$

puis sous la *deuxième* somme le terme $x_i$ ne varie pas donc il en sort :

$$w_n = \sum_{i=0}^{n} x_i \left( \sum_{k=i}^{n} y_{k-i} z_{n-k} \right)$$

Cela ressemble déjà mieux à l'expression de $w'_n$ ; il reste en fait à faire le changement d'indice $j = k - i$ sous la *deuxième* somme (la première ne bouge plus), on aura alors les termes $y_j$ et (comme $k = j + i$) $z_{n-k} = z_{n-j-i}$. De plus si $k = i$ alors $j = 0$ et si $k = n$ alors $j = n - i$. Donc on trouve

$$w_n = \sum_{i=0}^{n} x_i \left( \sum_{j=0}^{n-i} y_j z_{n-j-i} \right)$$

et ceci est bien la même chose que $w'_n$, quitte à renommer sous la deuxième somme la variable muette $j$ en $k$. □

9. Le produit de convolution admet-il un élément neutre ?

*Correction.* La question 4 montre précisément que la suite $(\varepsilon_n)_{n\in\mathbb{N}}$ est un élément neutre pour le produit de convolution : pour toute suite $(u_n)_{n\in\mathbb{N}}$ le produit de convolution de $(u_n)_{n\in\mathbb{N}}$ avec $(\varepsilon_n)_{n\in\mathbb{N}}$ est égal en tant que suite à $(u_n)_{n\in\mathbb{N}}$. □

# Informatique sujet A

Corrigeons uniquement la partie informatique du sujet. Cela ressemble beaucoup à ce que nous avons déjà fait avec des tuples, mais cette fois avec des listes.

1. (a) Rappelons que si $z = a + ib$ correspond à la liste `L` alors on récupère les parties réelles et imaginaires via `a = L[0]` et `b = L[1]`, et le conjugué est $a - ib$ :

```
def conjugue(L):
    a = L[0]
    b = L[1]
    return [a, -b]
```

   (b) Idem. Si $z = a + ib$ et $w = c + id$ alors la somme est $z + w = (a + c) + i(b + d)$ :

```
def somme(L, M):
    a = L[0]
    b = L[1]
    c = M[0]
    d = M[1]
    return [a+c, b+d]
```

   (c) Idem, et le produit est $z \times w = (ad - bc) + i(ac + bd)$ :

```
def produit(L, M):
    a = L[0]
    b = L[1]
    c = M[0]
    d = M[1]
    return [a*c - b*d, a*d + b*c]
```

   (d) On s'inspire de la façon la plus habituelle possible : au départ une variable `z` contient le premier terme de la suite, qui est ici le nombre complexe $1 + i$ donc la liste `[1, 1]`. Puis dans une boucle à répéter $n$ fois on fait `z = f(z)` où `f` est la fonction qui permet de passer d'un terme au suivant... à écrire calmement en composant dans le bon sens les fonctions précédentes et le nombres complexes qui apparaissent :

```
def suite(n):
    z = [1, 1]
    for i in range(n):
        # la variable z correspond à z_i
        z = somme(produit([-1, 9], z), produit([8, 4], conjugue(z)))
    # à la fin z représente z_n
    return z
```

   (e) Commentons la fonction

```
def mystere(): # c'est une fonction sans argument
    n=1 # pourquoi pas
    L=suite(n) # donc L correspond à z_1
    while L[0]!=L[1] # tant que les parties réelles et imaginaires de z sont différents
        # au début de boucle L représente z_n (c'est bien le cas avec n=1)
        n=n+1
        L=suite(n)
        # en fin de boucle L représente bien z_(n+1)
    return n
```

   La fonction boucle tant que les parties réelles et imaginaires de la suite $(z_n)_{n\in\mathbb{N}}$ sont différentes. Elle sort donc de la boucle à la première valeur de $n$ à laquelle elles sont égales. Plus précisément $n > 0$ car la boucle et la suite démarrent à 1 (en fait, $z_0$ vérifie cette condition, mais on démarre à 1).
   À ce stade il n'est pas démontré mathématiquement que la suite $(z_n)_{n\in\mathbb{N}}$ va atteindre une valeur $z_n$ pour laquelle $\mathrm{Re}(z_n) = \mathrm{Im}(z_n)$. Donc :
   — Soit il n'existe jamais de $n > 0$ tel que $\mathrm{Re}(z_n) = \mathrm{Im}(z_n)$, dans ce cas il s'agit d'une boucle infinie.
   — Soit il existe un tel entier et donc la fonction retourne le plus petit $n > 0$ tel que $\mathrm{Re}(z_n) = \mathrm{Im}(z_n)$.
   Remarque (complément) : telle quelle, la fonction `mystere` manque de sérieux ! En effet dans la boucle on appelle la fonction `suite(n)` à chaque fois, et la fonction `suite` elle-même reprend le calcul de la suite depuis le début ! Jamais un élève de BCPST1B n'oserait écrire une chose pareille ! Mais plutôt :

```
def mystereserieux():
    n = 1
    z = suite(n)
    while z[0] != z[1]:
        z = somme(produit([-1, 9], z), produit([8, 4], conjugue(z)))
        n = n + 1
    return n
```