Plaquette 5 – Problèmes de synthèse : seuils, dichotomie, simulations et intégrales

Algorithmique et Python — Terminale spé maths

Corrections détaillées

Boîte à outils

Boucle de seuil. Tant que la condition d'arrêt n'est pas atteinte, on avance d'un cran :

```
n = 0
u = u0
while u < seuil:
    u = f(u)
    n = n + 1
print(n)
```

La boucle **termine** si la suite est monotone et de limite dépassant le seuil.

Dichotomie pour une équation $f(x) = 0$ sur $[a\,;b]$ avec f continue et $f(a)f(b) < 0$: on coupe l'intervalle en deux et on garde la moitié où le **changement de signe** persiste. Après k étapes, l'amplitude vaut $\dfrac{b-a}{2^k}$.

Sommes de Riemann pour $\int_a^b f$, avec $h = \dfrac{b-a}{n}$:

$$R_n = h\sum_{k=0}^{n-1} f(a+kh) \qquad \text{(rectangles à gauche)}$$

$$T_n = h\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a+kh)\right) \qquad \text{(trapèzes)}$$

Module random.

- *random()* renvoie un flottant de $[0\,;1[$ uniformément.
- *randint(a, b)* renvoie un entier de $[\![a\,;b]\!]$, **bornes incluses**.
- Simuler un succès de probabilité p : tester si *random()* $< p$.

Monte-Carlo. Pour estimer une aire, on tire N points au hasard dans un rectangle d'aire A et l'on compte la proportion f de points sous la courbe : l'aire vaut environ $f \times A$. La précision évolue en $\dfrac{1}{\sqrt{N}}$.

Vocabulaire. *Terminaison* : l'algorithme s'arrête. *Correction* : il renvoie le bon résultat. *Coût* : nombre d'opérations — linéaire pour une boucle de seuil, **logarithmique** pour une dichotomie.

Correction de l'exercice 1 – Lecture d'une boucle de seuil

- **Suite construite** : $u_n = 100 \times 0{,}8^n$, décroissante de limite 0.
- **Condition d'arrêt** : la boucle s'arrête dès que $u_n \leq 1$, soit

$$100 \times 0{,}8^n \leq 1 \Leftrightarrow 0{,}8^n \leq 0{,}01$$

- **Logarithme** (avec $\ln 0{,}8 < 0$, sens inversé) :

$$n \geq \frac{\ln 0{,}01}{\ln 0{,}8} \approx \frac{-4{,}6052}{-0{,}22314} \approx 20{,}64$$

— **Réponse** : le script affiche 21.
— **Vérifications** : $100 \times 0{,}8^{20} \approx 1{,}1529 > 1$ (la boucle continue) et $100 \times 0{,}8^{21} \approx 0{,}9223 \leq 1$ ✓
— **Terminaison** : (u_n) est décroissante de limite 0, donc elle finit par passer sous 1 — l'algorithme s'arrête toujours.
— **Attention à l'ordre des instructions** : u est modifié **avant** l'incrémentation, mais les deux lignes sont dans le même tour de boucle, donc n compte exactement le nombre de multiplications effectuées ✓

Réponse. 21.

Correction de l'exercice 2 – Écrire une boucle de seuil

Script.

```
n = 0
```

```
u = 2.0
while u <= 200:
    u = 1.05 * u + 3
    n = n + 1
print(n)
```

— **Croissance** : $u_1 = 5{,}1$, $u_2 = 8{,}355$ — la suite est strictement croissante, donc la boucle termine.
— **Forme explicite** : en posant $v_n = u_n + 60$, on a $v_{n+1} = 1{,}05 v_n$ (car -60 est le point fixe : $1{,}05x + 3 = x$ donne $x = -60$).
— Donc $v_n = 62 \times 1{,}05^n$ et $u_n = 62 \times 1{,}05^n - 60$.
— **Résolution** : $u_n > 200$ équivaut à $1{,}05^n > \dfrac{260}{62} \approx 4{,}1935$, soit

$$n > \frac{\ln 4{,}1935}{\ln 1{,}05} \approx \frac{1{,}4337}{0{,}048790} \approx 29{,}39$$

— **Réponse** : $n = 30$.
— **Vérifications** : $u_{29} = 62 \times 1{,}05^{29} - 60 \approx 195{,}20$ et $u_{30} \approx 207{,}96 > 200$ ✓
— **Point de vigilance** : la condition de la boucle est $u <= 200$, complémentaire de l'objectif $u > 200$ — inverser la condition est l'erreur classique.

Réponse. $n = 30$ ($u_{30} \approx 208{,}0$).

Correction de l'exercice 3 – Dichotomie

— **Existence d'une solution** : $f(1) = -1 < 0$ et $f(2) = 7 > 0$, avec f continue — le théorème des valeurs intermédiaires garantit une racine dans $]1\,;2[$.
— **Unicité** : $f'(x) = 3x^2 + 1 > 0$, donc f est strictement croissante : la racine est **unique**.
— **Nombre d'étapes** : l'amplitude après k étapes vaut $\dfrac{1}{2^k}$, et l'on cherche

$$\frac{1}{2^k} \leq 0{,}001 \Leftrightarrow 2^k \geq 1000$$

— Comme $2^9 = 512$ et $2^{10} = 1024$, il faut **10 étapes**.
— **Encadrement final** : la racine vaut $\alpha \approx 1{,}2134$, et le script affiche un intervalle d'amplitude $\dfrac{1}{1024} \approx 0{,}000977$.

— **Contrôle** : $f(1{,}213) \approx -0{,}002$ et $f(1{,}214) \approx 0{,}0032$ — le changement de signe a bien lieu là ✓
— **Choix du test** : *f(a) \| f(m) <= 0 garde la moitié gauche quand le signe change entre a et m. Écrire* f(m) > 0 *fonctionnerait aussi ici, mais uniquement parce que f est croissante — le test par le produit est général*.

Réponse. 10 étapes ; $\alpha \approx 1{,}213$.

Correction de l'exercice 4 – Somme de Riemann

Script.

```
n = 100
```

```
h = 1 / n
s = 0
for k in range(n):
    s = s + (k * h) ** 2
print(s * h)
```

— **Valeur exacte** : $\int_0^1 x^2\,\mathrm{d}x = \left[\frac{x^3}{3}\right]_0^1 = \frac{1}{3} \approx 0{,}3333$.
— **Valeur calculée** : la somme vaut exactement

$$R_n = \frac{1}{n^3}\sum_{k=0}^{n-1} k^2 = \frac{(n-1)n(2n-1)}{6n^3}$$

— **Pour** $n = 100$: $R_{100} = \frac{99 \times 100 \times 199}{6 \times 10^6} = \frac{1\,970\,100}{6\,000\,000} = 0{,}32835$.
— **Erreur** : $0{,}33333 - 0{,}32835 = 0{,}00498$, soit environ 1,5 %.
— **Sens de l'erreur** : $x \mapsto x^2$ est **croissante**, donc les rectangles à gauche **sous-estiment** l'intégrale ✓
— **Amélioration par les trapèzes** : $T_{100} = R_{100} + \frac{h}{2}(f(1) - f(0)) = 0{,}32835 + 0{,}005 = 0{,}33335$ — erreur ramenée à 2×10^{-5}, soit 250 fois mieux.
— **Vitesse de convergence** : l'erreur des rectangles est en $\frac{1}{n}$, celle des trapèzes en $\frac{1}{n^2}$.

Réponse. $R_{100} = 0{,}32835$ contre $\frac{1}{3} \approx 0{,}33333$ (sous-estimation).

Correction de l'exercice 5 – Simulation d'une loi de Bernoulli

Script.

```
from random import random
```

```
def binomiale(n, p):
    s = 0
    for i in range(n):
        if random() < p:
            s = s + 1
    return s
```

— **Justification du test** : *random()* est uniforme sur $[0\,;1[$, donc $P(\text{random}() < p) = p$ ✓
— **Loi simulée** : les épreuves sont **identiques et indépendantes**, donc *binomiale(n, p)* suit la loi $B(n\,;p)$.
— **Contrôle empirique** : sur 10 000 appels de *binomiale(20, 0.3)*, la moyenne doit approcher $E(X) = 6$ et l'écart type $\sqrt{20 \times 0{,}3 \times 0{,}7} = \sqrt{4{,}2} \approx 2{,}049$.

— **Erreur fréquente** : écrire *random() <= p* change la probabilité de façon négligeable (l'événement *random() == p* est de probabilité nulle), mais *randint(0, 1)* ne simulerait qu'un $p = 0{,}5$.
— **Autre piège** : réinitialiser s à l'intérieur de la boucle remettrait le compteur à zéro à chaque tour — le résultat ne vaudrait alors que 0 ou 1.
— **Coût** : n tirages, donc une complexité **linéaire** en n.

Réponse. La fonction simule la loi binomiale $B(n\,;p)$.

Correction de l'exercice 6 – Monte-Carlo et estimation de π

— **Principe** : on tire un point uniformément dans le carré $[0\,;1]^2$, d'aire 1.
— **Probabilité de tomber dans le quart de disque** : $\dfrac{\text{aire du quart de disque}}{\text{aire du carré}} = \dfrac{\pi/4}{1} = \dfrac{\pi}{4}$.
— **Estimation** : la fréquence $\frac{c}{N}$ approche $\frac{\pi}{4}$, donc $4\frac{c}{N}$ approche π ✓
— **Loi de** c : $c \sim B\left(N\,;\frac{\pi}{4}\right)$, avec $\frac{\pi}{4} \approx 0{,}7854$.
— **Écart type de la fréquence** : $\sqrt{\dfrac{0{,}7854 \times 0{,}2146}{10^6}} \approx 4{,}106 \times 10^{-4}$.
— **Écart type de l'estimation** : $4 \times 4{,}106 \times 10^{-4} \approx 1{,}64 \times 10^{-3}$.
— **Conclusion** : l'erreur typique est de l'ordre de 0,002 — on obtient environ **deux à trois décimales** de π avec un million de tirages.
— **Coût de la précision** : gagner une décimale exige de diviser l'erreur par 10, donc de multiplier N par 100. Atteindre 10^{-6} demanderait $N \approx 10^{12}$ tirages : Monte-Carlo est **robuste** mais très **lent**.
— **Comparaison** : la méthode des trapèzes sur $\int_0^1 \sqrt{1-x^2}\,\mathrm{d}x$ atteint la même précision avec quelques centaines de points seulement.

Réponse. Fréquence $\to \frac{\pi}{4}$; erreur typique $\approx 1{,}6 \times 10^{-3}$ pour $N = 10^6$.

Correction de l'exercice 7 – Recherche du maximum d'une liste

Script.

```
def maximum(L):
    m = L[0]
    for x in L:
        if x > m:
            m = x
    return m
```

— **Initialisation** : on part du **premier** élément, jamais de 0 — sinon une liste de nombres tous négatifs renverrait 0, ce qui est faux.
— **Invariant de boucle** : après chaque tour, m est le maximum des éléments déjà parcourus. À la fin, c'est le maximum de toute la liste ✓
— **Terminaison** : la boucle *for* parcourt un nombre fini d'éléments, elle s'arrête toujours.
— **Coût** : une comparaison par élément, soit n comparaisons — complexité **linéaire**.
— **Optimalité** : on ne peut pas faire mieux. Tout élément non examiné pourrait être le maximum, donc il faut au minimum n lectures.
— **Cas limite** : une liste **vide** provoque une erreur sur *L[0]* — il faudrait le traiter explicitement selon l'usage voulu.

Réponse. Parcours avec mémorisation du plus grand ; coût linéaire (n comparaisons).

Correction de l'exercice 8 – Suite de Syracuse

Script.

```
def syracuse(u):
    n = 0
    while u != 1:
        if u % 2 == 0:
            u = u // 2
        else:
            u = 3 * u + 1
        n = n + 1
    return n
```

— **Trajectoire pour** $u_0 = 7$:
$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$.

— **Décompte** : 16 étapes.

— **Altitude maximale** : 52 — la suite **monte** avant de descendre, ce qui interdit tout argument de décroissance simple.

— **Terminaison** : elle n'est **pas démontrée** en général ! La conjecture de Syracuse (Collatz), ouverte depuis 1937, affirme que la suite atteint toujours 1. Elle est vérifiée par ordinateur jusqu'à 2^{68} environ.

— **Détails techniques** : *//* est la division **entière** (indispensable, sinon u devient un flottant et le test *u != 1* peut ne jamais être satisfait) ; *%* donne le reste.

— **Autres valeurs** : $u_0 = 6$ donne 8 étapes, $u_0 = 27$ en donne 111 avec un maximum de 9232 — la longueur du vol est très irrégulière.

Réponse. 16 étapes pour $u_0 = 7$ (maximum atteint : 52).

Correction de l'exercice 9 – Calcul d'une somme

— **Somme calculée** : $\sum_{k=1}^{10} \dfrac{1}{k(k+1)}$.

— **Décomposition télescopique** :

$$\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}$$

— **Somme** :

$$\sum_{k=1}^{10} \left(\frac{1}{k} - \frac{1}{k+1} \right) = 1 - \frac{1}{11} = \frac{10}{11} \approx 0{,}909091$$

— **Contrôle des bornes** : *range(1, 11)* parcourt $k = 1$ à $k = 10$ — la borne supérieure est **exclue**, piège classique.

— **Vérification des premiers termes** :
$\frac{1}{2} + \frac{1}{6} + \frac{1}{12} = 0{,}5 + 0{,}16667 + 0{,}08333 = 0{,}75 = 1 - \frac{1}{4}$ ✓

— **Généralisation** : pour n termes, la somme vaut $1 - \dfrac{1}{n+1}$, de limite 1 — la série converge.

— **Remarque numérique** : le calcul flottant donne 0,9090909090909091, à comparer à $\frac{10}{11}$ exact. L'écart relatif est de l'ordre de 10^{-16}, la précision machine.

Réponse. $\dfrac{10}{11} \approx 0{,}909091$.

Correction de l'exercice 10 – Simulation d'une marche aléatoire

Script.

```
from random import randint

def marche(n):
    x = 0
    for i in range(n):
        if randint(0, 1) == 1:
            x = x + 1
        else:
            x = x - 1
    return x
```

— **Modèle** : $X = \sum_{i=1}^{n} \varepsilon_i$, où chaque ε_i vaut ± 1 avec la probabilité $\frac{1}{2}$.

— **Espérance d'un pas** : $1 \times 0{,}5 - 1 \times 0{,}5 = 0$, donc par linéarité

$$E(X) = 0$$

— **Variance d'un pas** : $E\left(\varepsilon^2\right) - 0 = 1$, donc par indépendance $V(X) = n$.

— **Écart type pour** $n = 100$: $\sigma(X) = \sqrt{100} = 10$.

— **Lecture** : la marche revient en moyenne à son point de départ, mais s'en écarte typiquement de 10 pas après 100 pas — c'est la fameuse loi en $\sqrt{n}$.

— **Parité** : après 100 pas, la position est **toujours paire** (elle vaut $2k - 100$ où k est le nombre de pas montants). Elle ne peut donc jamais valoir 1 ou 3.

— **Lien avec la binomiale** : si k est le nombre de pas $+1$, alors $k \sim B(100\,;0{,}5)$ et $X = 2k - 100$, ce qui redonne $E(X) = 2 \times 50 - 100 = 0$ et $V(X) = 4 \times 25 = 100$ ✓

Réponse. $E(X) = 0$ et $\sigma(X) = 10$ pour $n = 100$.

Correction de l'exercice 11 – Méthode d'Euler

Script.

```
n = 10
h = 1 / n
y = 1.0
for k in range(n):
    y = y + h * y
print(y)
```

— **Principe** : l'approximation affine $y(t + h) \approx y(t) + hy'(t) = y(t) + hy(t)$, soit $y_{k+1} = (1 + h)y_k$.

— **Forme explicite** : $y_n = (1 + h)^n = 1{,}1^{10}$.

— **Valeur calculée** :

$$1{,}1^{10} \approx 2{,}59374$$

— **Valeur exacte** : $y(1) = e \approx 2{,}71828$.

— **Erreur** : 0,12454, soit environ 4,6 % — la méthode d'Euler **sous-estime** ici, car la fonction est convexe et la tangente reste en dessous de la courbe.

— **Convergence** : avec $n = 100$, on obtient $1{,}01^{100} \approx 2{,}70481$ (erreur 0,0135) ; avec $n = 1000$, $1{,}001^{1000} \approx 2{,}71692$ (erreur 0,00136).

— **Vitesse** : l'erreur est divisée par 10 quand n est multiplié par 10 — la méthode est d'**ordre 1**.

— **Limite théorique** : $\lim_{n\to+\infty}\left(1+\frac{1}{n}\right)^n = e$ — c'est exactement la définition historique de e ✓

Réponse. $1{,}1^{10} \approx 2{,}5937$ contre $e \approx 2{,}7183$ (erreur 4,6 %).

Correction de l'exercice 12 – Test de primalité

Script.

```
def est_premier(n):
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False
        d = d + 1
    return True
```

— **Justification de la borne** : si $n = ab$ avec $1 < a \leq b$, alors $a^2 \leq ab = n$, donc $a \leq \sqrt{n}$.
— **Conclusion** : tout nombre composé admet un diviseur $\leq \sqrt{n}$ — il est inutile de chercher au-delà ✓
— **Écriture du test** : *d \ d <= n** évite le calcul d'une racine carrée flottante et ses arrondis, source d'erreurs sur les grands entiers.
— **Coût** : au plus $\sqrt{n}$ divisions, contre n pour une boucle naïve. Pour $n = 10^{12}$, cela fait 10^6 opérations au lieu de 10^{12} — un gain d'un **facteur un million**.
— **Tests** : *est_premier(97)* renvoie *True* (on teste d de 2 à 9, car $10^2 = 100 > 97$) ; *est_premier(91)* renvoie *False* car $91 = 7 \times 13$.
— **Optimisation supplémentaire** : après avoir testé 2, on peut n'essayer que les d impairs, ce qui divise encore le coût par deux.

Réponse. Tout composé a un diviseur $\leq \sqrt{n}$; coût en $\sqrt{n}$ au lieu de n.

Correction de l'exercice 13 – Simulation et fréquence

— **Probabilité théorique** : 6 couples sur 36 donnent une somme de 7, soit $p = \frac{1}{6} \approx 0{,}16667$.
— **Fréquence attendue** : environ 0,167, donc $c \approx 1667$.
— **Loi de** c : $c \sim B\left(10\,000\,;\frac{1}{6}\right)$, d'où $E(c) = 1666{,}67$ et

$$\sigma(c) = \sqrt{10\,000 \times \frac{1}{6} \times \frac{5}{6}} = \sqrt{1388{,}89} \approx 37{,}27$$

— **Écart type de la fréquence** : $\frac{37{,}27}{10\,000} \approx 0{,}00373$.
— **Intervalle typique** ($\pm 2\sigma$) : la fréquence affichée sera presque toujours dans $[0{,}1592\,;0{,}1741]$.
— **Attention à* randint* **: les bornes sont** incluses*, *donc* randint(1, 6) *donne bien un dé à six faces. Avec* randint(0, 6)* on simulerait un dé à sept faces, biaisant tout le calcul.
— **Erreur à éviter** : appeler *randint(1, 6)* une seule fois et le doubler simulerait $2X$ (toujours pair), pas la somme de deux dés indépendants.
— **Contrôle par Tchebychev** : $P\left(|F - \frac{1}{6}| \geq 0{,}01\right) \leq \frac{5/36}{10\,000 \times 0{,}0001} \approx 0{,}139$.

Réponse. Fréquence $\approx 0{,}167$; $\sigma(c) \approx 37{,}3$, soit 0,0037 sur la fréquence.

Correction de l'exercice 14 – Recherche par balayage

Script.

```
from math import exp

x = 0.0
while exp(x) - 3 * x > 0:
    x = x + 0.01
print(x - 0.01, x)
```

— **Existence** : posons $g(x) = e^x - 3x$. Alors $g(0) = 1 > 0$ et $g(1) = e - 3 \approx -0{,}2817 < 0$.
— g est continue, donc le théorème des valeurs intermédiaires donne une racine dans $]0\,;1[$.
— **Unicité sur cet intervalle** : $g'(x) = e^x - 3 < 0$ pour $x < \ln 3 \approx 1{,}0986$, donc g est **strictement décroissante** sur $[0\,;1]$ ✓
— **Résultat** : la boucle s'arrête pour $x = 0{,}62$, car $g(0{,}61) \approx 0{,}0104 > 0$ et $g(0{,}62) \approx -0{,}0011 < 0$.
— **Encadrement** : $0{,}61 < \alpha < 0{,}62$, et plus précisément $\alpha \approx 0{,}6190$.
— **Vérification** : $e^{0{,}619} \approx 1{,}8571$ et $3 \times 0{,}619 = 1{,}857$ ✓
— **Coût** : environ 62 itérations pour une précision de 10^{-2}. Une **dichotomie** atteindrait 10^{-2} en 7 étapes seulement — le balayage est simple mais nettement moins efficace.
— **Seconde solution** : l'équation admet aussi une racine vers $x \approx 1{,}512$, hors de l'intervalle étudié.

Réponse. $0{,}61 < \alpha < 0{,}62$ (avec $\alpha \approx 0{,}619$).

Correction de l'exercice 15 – Comparaison de deux algorithmes

Boucle while — pour atteindre $u_n < 10^{-6}$:

```
n = 0
```

```
u = 1.0
while u >= 1e-6:
    u = u * 0.9
    n = n + 1
print(n)
```

— **Nombre d'itérations** : $0{,}9^n < 10^{-6}$ donne $n > \dfrac{-13{,}8155}{-0{,}10536} \approx 131{,}1$, donc $n = 132$.
— **Coût** : 132 multiplications.

Calcul direct par logarithme.

$$n = \left\lceil \frac{\ln 10^{-6}}{\ln 0{,}9} \right\rceil = \lceil 131{,}1 \rceil = 132$$

— **Coût** : deux logarithmes et une division — **indépendant** du seuil.

Comparaison.

— Le coût de la boucle est **proportionnel** à n, donc au logarithme du seuil : viser 10^{-12} demanderait 263 itérations.
— Le calcul direct reste à coût **constant**, quel que soit le seuil.
— **Vérifications** : $0{,}9^{131} \approx 1{,}0134 \times 10^{-6} \geq 10^{-6}$ et $0{,}9^{132} \approx 9{,}120 \times 10^{-7} < 10^{-6}$ ✓
— **Intérêt pédagogique de la boucle** : elle fonctionne même quand la suite n'a **pas** de forme explicite (suite définie par récurrence non géométrique), là où le logarithme est inutilisable. C'est son vrai domaine d'emploi.

Réponse. $n = 132$ dans les deux cas ; boucle en $O(n)$, logarithme à coût constant.

Correction de l'exercice 16 – Synthèse : intégrale et suite

a) **Script.**

```
from math import exp

def integrale(n, N=100):
    h = 1 / N
    s = (0 + exp(-1)) / 2
    for k in range(1, N):
        x = k * h
        s = s + x**n * exp(-x)
    return s * h
```

— Les termes de bord sont $f(0) = 0$ (pour $n \geq 1$) et $f(1) = e^{-1} \approx 0{,}36788$.
— **Valeurs obtenues** : $I_1 \approx 0{,}26423$, $I_2 \approx 0{,}16061$, $I_5 \approx 0{,}07131$.
— **Contrôle exact** : $I_1 = 1 - \dfrac{2}{e} \approx 0{,}264241$ ✓ (calcul par intégration par parties).

b) **Positivité** : sur $[0\,;1]$, $x^n e^{-x} \geq 0$, donc $I_n \geq 0$.

Décroissance : pour $x \in [0\,;1]$, on a $x^{n+1} \leq x^n$, donc

$$I_{n+1} - I_n = \int_0^1 x^n (x-1) e^{-x}\,\mathrm{d}x \leq 0$$

car $x - 1 \leq 0$ sur $[0\,;1]$. La suite est donc **décroissante** ✓

c) **Encadrement** : sur $[0\,;1]$, on a $e^{-1} \leq e^{-x} \leq 1$, donc en multipliant par $x^n \geq 0$ et en intégrant

$$\frac{1}{e(n+1)} \leq I_n \leq \frac{1}{n+1}$$

— En effet $\int_0^1 x^n\,\mathrm{d}x = \dfrac{1}{n+1}$, et le minorant vaut $e^{-1} \times \dfrac{1}{n+1}$.
— **Limite** : les deux bornes tendent vers 0, donc par encadrement

$$\lim_{n \to +\infty} I_n = 0$$

— **Contrôle numérique pour** $n = 5$: l'encadrement donne $[0{,}06131\,;0{,}16667]$, et le script renvoie $I_5 \approx 0{,}07131$ ✓ — bien à l'intérieur.
— **Contrôle pour** $n = 1$: $[0{,}18394\,;0{,}5]$ contient $0{,}26424$ ✓
— **Finesse de l'encadrement** : le rapport des deux bornes vaut $e \approx 2{,}72$ — l'encadrement donne le bon **ordre de grandeur** mais pas la valeur. Pour n grand, $I_n \sim \dfrac{1}{e} \cdot \dfrac{1}{n}$, car la masse de x^n se concentre près de $x = 1$ où $e^{-x} \approx e^{-1}$.
— **Vérification de cette équivalence** : nI_n vaut $0{,}3565$ pour $n = 5$ et tend vers $e^{-1} \approx 0{,}3679$ ✓
— **Leçon** : un calcul numérique qui **contredit** un encadrement démontré signale une erreur, de méthode ou de calcul — c'est un excellent test de cohérence à pratiquer systématiquement.

Réponse. b) décroissante et positive · c) $\dfrac{1}{e(n+1)} \leq I_n \leq \dfrac{1}{n+1}$, donc $I_n \to 0$.