Plaquette 5 – Problèmes de synthèse : algorithmes du programme et débogage

Algorithmique et Python — Première spé maths

Corrections détaillées

Boîte à outils

Les algorithmes à connaître en première.

Algorithme	Principe	Coût
seuil	boucle *while* jusqu'au dépassement	$O(n)$
dichotomie	couper l'intervalle en deux	$O(\log n)$
somme de termes	accumulateur dans une boucle *for*	$O(n)$
recherche d'extremum	balayage avec mémorisation	$O(n)$
simulation	*random* répété, puis fréquence	$O(N)$
méthode d'Euler	$y_{k+1} = y_k + hf(x_k, y_k)$	$O(n)$

Terminaison. On exhibe une quantité qui **décroît strictement** vers une borne (l'amplitude en dichotomie, l'écart au seuil pour une suite monotone), ou l'on invoque la finitude d'une boucle *for*.

Correction. On énonce un **invariant** : une propriété vraie avant la boucle, préservée à chaque tour, et qui donne le résultat à la sortie.

Exemple d'invariant — pour la recherche du maximum : « après k tours, m est le maximum des k premiers éléments ».

Méthode d'Euler pour $y' = f(x, y)$ avec $y(x_0) = y_0$:

```
def euler(x0, y0, h, n):
    x, y = x0, y0
    for k in range(n):
        y = y + h * f(x, y)
        x = x + h
    return y
```

L'erreur est d'ordre h : diviser le pas par 10 divise l'erreur par 10.

Débogage : les cinq suspects.

1. Accumulateur initialisé **dans** la boucle. 2. Bornes de *range* décalées d'une unité. 3. Inégalité stricte au lieu de large (ou l'inverse). 4. `//` et `/` confondus. 5. Boucle *while* dont la condition ne peut jamais devenir fausse.

Correction de l'exercice 1 – Méthode d'Euler

— **Récurrence** : $y_{k+1} = y_k + 0{,}25 \times 2y_k = 1{,}5y_k$.
— **Valeurs** : $y_0 = 1$; $y_1 = 1{,}5$; $y_2 = 2{,}25$; $y_3 = 3{,}375$; $y_4 = 5{,}0625$ ✓
— **Forme explicite** : $y_n = 1{,}5^n$, donc $y(1) \approx 5{,}0625$.
— **Solution exacte** : $y(x) = e^{2x}$, donc $y(1) = e^2 \approx 7{,}38906$ ✓
— **Erreur** : 2,32656, soit une erreur relative de 31,5 % — considérable ✓
— **Sens de l'erreur** : la méthode **sous-estime**, car la fonction est convexe et la tangente reste en dessous ✓

— **Avec** $h = 0{,}1$: $y(1) \approx 1{,}2^{10} \approx 6{,}19174$, erreur de 1,19732 (16,2 %) ✓
— **Avec** $h = 0{,}01$: $1{,}02^{100} \approx 7{,}24465$, erreur de 0,14441 (2,0 %) ✓
— **Ordre de convergence** : diviser h par 10 divise l'erreur par environ 10 — méthode d'**ordre 1** ✓
— **Contrôle** : $\dfrac{2{,}3266}{1{,}1973} \approx 1{,}94$ et $\dfrac{1{,}1973}{0{,}1444} \approx 8{,}29$ — le facteur approche 10 à mesure que h diminue ✓
— **Limite théorique** : $\lim_{n \to +\infty} \left(1 + \dfrac{2}{n}\right)^n = e^2$ ✓

Réponse. $y(1) \approx 5{,}0625$ contre $e^2 \approx 7{,}389$ — erreur de 31,5 % avec $h = 0{,}25$.

Correction de l'exercice 2 – Déboguer un calcul de somme

— **Erreur fatale** : *range(n)* commence à $k = 0$, donc *1 / 0* provoque une **division par zéro** ✓
— **Erreur de borne** : même corrigée, la boucle s'arrêterait à $n - 1$ au lieu de n ✓
— **Script corrigé** :

```
def harmonique(n):
    s = 0
    for k in range(1, n + 1):
        s = s + 1 / k
    return s
```

— **Contrôles** : *harmonique(1)* = 1 ✓ ; *harmonique(2)* = 1,5 ✓ ; *harmonique(3)* ≈ 1,83333 ✓
— **Valeur pour** $n = 10$: ≈ 2,92897 ✓
— **Approximation** : $H_n \approx \ln n + 0{,}5772$ — pour $n = 10$, cela donne 2,87979 ✓
— **Écart** : 0,049, qui décroît comme $\dfrac{1}{2n}$ ✓
— **Contrôle** : $\dfrac{1}{20} = 0{,}05$ ✓
— **Divergence** : $H_n \to +\infty$, mais très lentement — il faut $n \approx 12\,367$ pour atteindre 10 ✓
— **Leçon de débogage** : tester systématiquement sur $n = 1$, le plus petit cas — ici, l'erreur serait apparue immédiatement ✓
— **Piège du flottant** : *1 / k* donne un flottant ; avec *1 // k*, on obtiendrait 1 puis 0 partout, soit la valeur 1 pour tout n ✓

Réponse. *range(n)* provoque une division par zéro et s'arrête trop tôt : il faut *range(1, n + 1)*.

Correction de l'exercice 3 – Dichotomie avec compteur

Script.

```
a, b = 2.0, 3.0
n = 0
while b - a > 1e-4:
    m = (a + b) / 2
    if m*m < 7:
        a = m
    else:
        b = m
    n = n + 1
print(n, a, b)
```

— **Validité** : $2^2 = 4 < 7$ et $3^2 = 9 > 7$ ✓
— **Nombre d'étapes** : $\dfrac{1}{2^k} \leq 10^{-4}$ donne $2^k \geq 10\,000$.
— **Puissances** : $2^{13} = 8192$ et $2^{14} = 16\,384$, donc **quatorze** étapes ✓
— **Valeur exacte** : $\sqrt{7} \approx 2{,}6457513$ ✓
— **Encadrement final** : d'amplitude $\dfrac{1}{16\,384} \approx 6{,}1 \times 10^{-5}$ ✓
— **Trace** : $m = 2{,}5$ ($6{,}25 < 7$, on garde $[2{,}5\,;3]$) ; $m = 2{,}75$ ($7{,}5625 > 7$, on garde $[2{,}5\,;2{,}75]$) ; $m = 2{,}625$ ($6{,}890625 < 7$, on garde $[2{,}625\,;2{,}75]$) ✓
— **Terminaison** : l'amplitude est divisée par 2 à chaque tour, donc elle tend vers 0 ✓
— **Invariant de correction** : à chaque étape, $a^2 < 7 \leq b^2$ — la racine est donc toujours dans $[a\,;b]$ ✓
— **Coût par décimale** : environ 3,32 étapes ✓
— **Comparaison avec Newton** : $x_{n+1} = \dfrac{x_n}{2} + \dfrac{3{,}5}{x_n}$ atteindrait 10^{-4} en **trois** étapes depuis $x_0 = 3$ ✓
— **Contrôle Newton** : $x_1 \approx 2{,}66667$; $x_2 \approx 2{,}64583$; $x_3 \approx 2{,}6457513$ ✓

Réponse. 14 étapes, encadrant $\sqrt{7} \approx 2{,}64575$.

Correction de l'exercice 4 – Algorithme de seuil sur un modèle

Script.

```
v = 12000.0
```

```
n = 0
while v >= 9000:
    v = 0.92 * v + 600
    n = n + 1
print(n)
```

— **Point fixe** : $0{,}92x + 600 = x$ donne $0{,}08x = 600$, soit $x = 7500$ ✓
— **Forme explicite** : $v_n = 7500 + 4500 \times 0{,}92^n$ ✓
— **Contrôles** : $v_0 = 12\,000$ ✓ ; $v_1 = 7500 + 4140 = 11\,640$, et directement $11\,040 + 600 = 11\,640$ ✓
— **Terminaison** : la suite **décroît** vers $7500 < 9000$, donc le seuil est nécessairement franchi ✓
— **Résolution** : $7500 + 4500 \times 0{,}92^n < 9000$ donne $0{,}92^n < \dfrac{1}{3}$, soit

$$n > \frac{\ln(1/3)}{\ln 0{,}92} \approx \frac{-1{,}0986}{-0{,}083382} \approx 13{,}176$$

— **Réponse** : la 14^{e} année ✓
— **Vérifications** : $v_{13} = 7500 + 4500 \times 0{,}92^{13} \approx 7500 + 1522{,}1 = 9022{,}1 \geq 9000$ ✓
— $v_{14} \approx 7500 + 1400{,}4 = 8900{,}4 < 9000$ ✓
— **Avertissement** : un seuil de 7000 ne serait **jamais** atteint (la suite est minorée par 7500) et la boucle tournerait indéfiniment ✓
— **Interprétation** : le lac se stabilise là où les pertes compensent les apports : $0{,}08 \times 7500 = 600$ ✓
— **Temps de demi-approche** : l'écart au palier est divisé par 2 tous les $\dfrac{\ln 0{,}5}{\ln 0{,}92} \approx 8{,}313$ ans ✓

Réponse. La 14^{e} année ; terminaison car $v_n \to 7500 < 9000$ en décroissant.

Correction de l'exercice 5 – Invariant de boucle

— **Invariant** : après le tour d'indice k, on a $p = k!$ et $s = 1! + 2! + \cdots + k!$ ✓
— **Vérification initiale** : avant la boucle, $p = 1 = 0!$ et $s = 0$ (somme vide) ✓
— **Préservation** : si $p = (k-1)!$ au début du tour k, alors p devient $(k-1)! \times k = k!$, puis s reçoit $k!$ ✓
— **Conclusion à la sortie** : après $k = 5$, on a $s = 1! + 2! + 3! + 4! + 5!$ ✓
— **Valeurs de** p : 1 ; 2 ; 6 ; 24 ; 120 ✓
— **Valeurs de** s : 1 ; 3 ; 9 ; 33 ; 153 ✓
— **Résultat** : le script affiche 153 ✓
— **Contrôle** : $1 + 2 + 6 + 24 + 120 = 153$ ✓
— **Efficacité de la construction** : on calcule chaque factorielle à partir de la précédente, en **une** multiplication — au lieu de k multiplications par factorielle ✓
— **Coût total** : 5 multiplications et 5 additions, contre 15 multiplications pour une version naïve ✓
— **Ordre des instructions crucial** : mettre $s = s + p$ **avant** $p = p$ k* donnerait $0! + 1! + \cdots + 4! = 1 + 1 + 2 + 6 + 24 = 34$ ✓
— **Extension** : pour $n = 10$, la somme vaut 4 037 913 — elle est dominée par le dernier terme $10! = 3\,628\,800$ ✓

Réponse. Invariant : $p = k!$ et $s = \sum_{i=1}^{k} i!$; le script affiche 153.

Correction de l'exercice 6 – Déboguer une simulation

— **Première erreur** : le test *random() < 0.6* simule $p = 0{,}6$ et non 0,4 ✓
— **Seconde erreur** : $s > 2$ estime $P(X \geq 3)$ et non $P(X \geq 2)$ ✓
— **Script corrigé** : remplacer par *random() < 0.4* et $s >= 2$ ✓
— **Valeur théorique cherchée** : $P(X \geq 2)$ pour $B(5\,;0{,}4)$.
— **Calcul** : $P(X = 0) = 0{,}6^5 = 0{,}07776$ et $P(X = 1) = 5 \times 0{,}4 \times 0{,}6^4 = 0{,}2592$.

$$P(X \geq 2) = 1 - 0{,}07776 - 0{,}2592 = 0{,}66304$$

— **Ce qu'affiche le script erroné** : $P(X \geq 3)$ pour $B(5\,;0{,}6)$.
— **Calcul** : $P(X = 3) = 10 \times 0{,}216 \times 0{,}16 = 0{,}3456$; $P(X = 4) = 5 \times 0{,}1296 \times 0{,}4 = 0{,}2592$; $P(X = 5) = 0{,}07776$.

$$P(X \geq 3) = 0{,}3456 + 0{,}2592 + 0{,}07776 = 0{,}68256$$

— **Coïncidence troublante** : 0,683 est **très proche** de 0,663 — les deux erreurs se compensent presque ✓
— **Écart** : 0,0195, soit $\dfrac{0{,}0195}{\sqrt{0{,}663 \times 0{,}337 / 10\,000}} \approx 4{,}12\,\sigma$ — détectable, mais il faut y regarder de près ✓
— **Leçon de débogage** : deux erreurs peuvent se **masquer** mutuellement. Il faut tester chaque brique séparément — par exemple vérifier que la fréquence de succès élémentaire vaut bien 0,4 ✓
— **Symétrie en cause** : pour $B(n\,;p)$, on a $P(X \geq k) = P(Y \leq n - k)$ avec $Y \sim B(n\,;1-p)$ — d'où la quasi-compensation ✓

Réponse. *0.6* au lieu de *0.4*, et $s > 2$ au lieu de $s >= 2$; la vraie valeur est 0,66304.

Correction de l'exercice 7 – Somme de termes d'une suite

Script.

```
s = 0
```

```
u = 3.0
for k in range(21):
    s = s + u
    u = 0.8 * u + 1
print(s)
```

— **Point fixe** : $0{,}8x + 1 = x$ donne $0{,}2x = 1$, soit $x = 5$ ✓
— **Forme explicite** : avec $v_n = u_n - 5$, on a $v_{n+1} = 0{,}8v_n$ et $v_0 = -2$, donc

$$u_n = 5 - 2 \times 0{,}8^n$$

— **Contrôles** : $u_0 = 3$ ✓ ; $u_1 = 5 - 1{,}6 = 3{,}4$, et directement $2{,}4 + 1 = 3{,}4$ ✓
— **Somme** :

$$S = \sum_{k=0}^{20} \left(5 - 2 \times 0{,}8^k\right) = 21 \times 5 - 2 \times \frac{1 - 0{,}8^{21}}{1 - 0{,}8}$$

— **Calcul** : $0{,}8^{21} \approx 0{,}0092234$, donc la somme géométrique vaut $\dfrac{0{,}9907766}{0{,}2} \approx 4{,}953883$.
— **Total** : $105 - 2 \times 4{,}953883 \approx 105 - 9{,}907766 = 95{,}092234$ ✓
— **Contrôle par les premiers termes** : $3 + 3{,}4 + 3{,}72 + 3{,}976 + 4{,}1808 = 18{,}2768$ ✓
— **Contrôle par la formule** :
$5 \times 5 - 2 \times \dfrac{1 - 0{,}8^5}{0{,}2} = 25 - 2 \times 3{,}36128 = 25 - 6{,}72256 = 18{,}27744$ ✓
— **Ordre des instructions** : on ajoute u **avant** de le faire évoluer, pour compter u_0 à u_{20} et non u_1 à u_{21} ✓
— **Comportement asymptotique** : la somme se comporte comme $5n$ pour n grand, puisque $u_n \to 5$ ✓
— **Contrôle** : $\dfrac{95{,}09}{21} \approx 4{,}528$, moyenne proche de 5 ✓

Réponse. $S \approx 95{,}092$ (avec $u_n = 5 - 2 \times 0{,}8^n$).

Correction de l’exercice 8 – Balayage pour un extremum

Script.

```
def f(x):
    return x * (4 - x*x)

xm = 0
for k in range(201):
    x = k * 0.01
    if f(x) > f(xm):
        xm = x
print(xm, f(xm))
```

— **Calcul exact** : $f(x) = 4x - x^3$, donc $f'(x) = 4 - 3x^2$.
— **Racine positive** : $x = \dfrac{2}{\sqrt{3}} \approx 1{,}1547$ ✓
— **Maximum exact** :

$$f\left(\frac{2}{\sqrt{3}}\right) = \frac{2}{\sqrt{3}}\left(4 - \frac{4}{3}\right) = \frac{2}{\sqrt{3}} \times \frac{8}{3} = \frac{16}{3\sqrt{3}} \approx 3{,}07920$$

— **Résultat du balayage** : le maximum est trouvé en $x = 1{,}15$, avec $f(1{,}15) \approx 3{,}07913$ ✓
— **Erreur sur** x : 0,0047, inférieure au pas ✓

— **Erreur sur** f : seulement 7×10^{-5} — bien **plus petite** que l'erreur sur x ✓
— **Explication** : au voisinage d'un maximum, la fonction est **plate** ($f' = 0$), donc l'erreur sur f est d'ordre $(\Delta x)^2$ ✓
— **Contrôle** : $(0{,}0047)^2 \approx 2{,}2 \times 10^{-5}$, du bon ordre de grandeur ✓
— **Conséquence pratique** : un balayage grossier donne une **mauvaise** abscisse mais une **bonne** valeur du maximum ✓
— **Nombre de points** : *range(201)* couvre x de 0 à 2 inclus ✓
— **Piège du flottant** : $k \times 0{,}01$ n'est pas toujours exact en binaire (0,07 devient 0,07000000000000007) — sans conséquence ici ✓

Réponse. Balayage : $x = 1{,}15$ et $f \approx 3{,}07913$; exact : $x = \dfrac{2}{\sqrt{3}} \approx 1{,}1547$ et $f = \dfrac{16}{3\sqrt{3}} \approx 3{,}07920$.

Correction de l'exercice 9 – Algorithme de dichotomie générique

Fonction.

```
def dicho(f, a, b, eps):
    while b - a > eps:
        m = (a + b) / 2
        if f(a) * f(m) <= 0:
            b = m
        else:
            a = m
    return (a + b) / 2
```

— **Test du signe par le produit** : *f(a)* f(m) <= 0 *est général*, il ne suppose pas que f soit croissante ✓
— **Application** : $f(1) = -1 < 0$ et $f(2) = 7 > 0$, donc une racine est dans $]1\,;2[$ ✓
— **Unicité** : $f'(x) = 3x^2 + 1 > 0$, donc f est strictement croissante ✓
— **Racine** : $\alpha \approx 1{,}21341$ ✓
— **Contrôle** : $f(1{,}21341) \approx 1{,}78659 + 1{,}21341 - 3 \approx 0{,}00000$ ✓
— **Étapes pour** $\varepsilon = 10^{-6}$: $2^k \geq 10^6$ donne $k = 20$ ✓
— **Retour du milieu** : renvoyer $\dfrac{a+b}{2}$ plutôt que a **divise l'erreur par deux** ✓
— **Généricité** : la même fonction résout $\cos x = x$ (racine $\approx 0{,}73909$ sur $[0\,;1]$) ou $2^x = 5$ (racine $\approx 2{,}32193$ sur $[2\,;3]$) ✓
— **Précondition** : il faut $f(a)f(b) < 0$ — sinon la fonction renvoie n'importe quoi sans prévenir ✓
— **Amélioration** : ajouter *assert f(a)* f(b) < 0* en début de fonction ✓
— **Limite de la méthode** : elle ne trouve **qu'une** racine, même s'il y en a plusieurs dans l'intervalle ✓

Réponse. Dichotomie générique par le test de signe ; racine $\alpha \approx 1{,}21341$.

Correction de l'exercice 10 – Coût comparé de deux algorithmes

Par le logarithme.

$$n > \frac{\ln 100}{\ln 1{,}02} \approx \frac{4{,}6052}{0{,}019803} \approx 232{,}55$$

— **Réponse** : $n = 233$ ✓
— **Vérifications** : $1{,}02^{232} \approx 98{,}9099$ et $1{,}02^{233} \approx 100{,}8881$ ✓

— **Coût** : deux logarithmes et une division — **constant** ✓

Par une boucle.

```
u = 1.0
```

```
n = 0
while u <= 100:
    u = 1.02 * u
    n = n + 1
print(n)
```

— **Coût** : 233 multiplications ✓

Par dichotomie sur n. On cherche le plus petit entier de $[\![0\,;1000]\!]$ vérifiant la condition.

— **Coût** : $\log_2 1000 \approx 10$ évaluations de $1{,}02^n$ ✓

— **Nuance** : chaque évaluation de $1{,}02^n$ coûte elle-même $O(\log n)$ multiplications par exponentiation rapide ✓

— **Classement** : logarithme ($O(1)$) < dichotomie ($O(\log^2 n)$) < boucle ($O(n)$) ✓

— **En pratique** : pour $n = 233$, les trois méthodes sont instantanées ; l'écart ne compte que pour des seuils gigantesques ✓

— **Pour un seuil de** 10^{100} : le logarithme reste instantané, la boucle ferait 11 628 tours ✓

— **Contrôle** : $\dfrac{\ln 10^{100}}{\ln 1{,}02} \approx \dfrac{230{,}26}{0{,}019803} \approx 11\,627{,}7$ ✓

— **Quand la boucle reste seule possible** : dès que la suite n'a pas de forme explicite ✓

Réponse. $n = 233$; logarithme en temps constant, dichotomie en ≈ 10 étapes, boucle en 233 tours.

Correction de l'exercice 11 – Déboguer une méthode d'Euler

— **Erreur** : $y' = -y$, donc la récurrence doit être $y_{k+1} = y_k + h \times (-y_k)$, soit $y = y$ - h y* ✓

— **Ce qu'affiche le script erroné** : $1{,}5^4 = 5{,}0625$ — une **croissance** au lieu d'une décroissance ✓

— **Détection immédiate** : la solution de $y' = -y$ **décroît** ; obtenir 5 au lieu de moins de 1 est absurde ✓

— **Script corrigé** : $y = y$ - h y*, soit $y_{k+1} = 0{,}5y_k$.

— **Valeurs** : 1 ; 0,5 ; 0,25 ; 0,125 ; 0,0625 ✓

— **Résultat** : $y(2) \approx 0{,}0625$.

— **Solution exacte** : $y(x) = e^{-x}$, donc $y(2) = e^{-2} \approx 0{,}135335$ ✓

— **Erreur** : 0,072835, soit 53,8 % — énorme, car le pas est très grossier ✓

— **Avec** $h = 0{,}1$: $y(2) = 0{,}9^{20} \approx 0{,}121577$, erreur de 0,013758 (10,2 %) ✓

— **Avec** $h = 0{,}01$: $0{,}99^{200} \approx 0{,}133980$, erreur de 0,001356 (1,0 %) ✓

— **Ordre 1 confirmé** : diviser h par 10 divise l'erreur par environ 10 ✓

— **Danger supplémentaire** : avec $h = 2$, on aurait $y_1 = 1 - 2 = -1$ — un résultat **négatif**, alors que $e^{-x} > 0$ toujours. La méthode d'Euler devient **instable** si h dépasse 1 ici ✓

— **Condition de stabilité** : il faut $|1 - h| < 1$, soit $0 < h < 2$ ✓

Réponse. Il faut $y = y$ - h y* ; on obtient alors $y(2) \approx 0{,}0625$ contre $e^{-2} \approx 0{,}1353$.

Correction de l'exercice 12 – Simulation et intervalle de fluctuation

Script.

```
from random import randint

c = 0
for k in range(200):
    s = 0
    for i in range(100):
        s = s + randint(0, 1)
    f = s / 100
    if f < 0.4 or f > 0.6:
        c = c + 1
print(c)
```

— **Intervalle en effectifs** : $40 \leq s \leq 60$ ✓
— **Probabilité de sortie** : $P(X \leq 39) + P(X \geq 61)$ pour $X \sim B(100\,;0{,}5)$.
— **Valeurs** : 0,017600 chacune, par symétrie, soit ≈ 0,035200 ✓
— **Comptage attendu** : $200 \times 0{,}0352 = 7{,}04$ échantillons ✓
— **Écart type** : $\sqrt{200 \times 0{,}0352 \times 0{,}9648} \approx 2{,}6069$ ✓
— **Intervalle typique** : $[1{,}8\,;12{,}3]$, soit entre 2 et 12 échantillons ✓
— **Conclusion** : on attend environ 7 échantillons hors intervalle ✓
— **Piège du** 5 % : l'intervalle $[0{,}4\,;0{,}6]$ n'est **pas** exactement l'intervalle à 95 % — il est plus large, d'où les 3,5 % au lieu de 5 % ✓
— **Astuce du script** : *randint(0, 1)* simule directement pile (1) ou face (0), et la somme donne le nombre de piles ✓
— **Coût** : $200 \times 100 = 20\,000$ tirages, instantané ✓
— **Contrôle complémentaire** : la moyenne des 200 fréquences doit approcher 0,5 à $\dfrac{0{,}05}{\sqrt{200}} \approx 0{,}0035$ près ✓

Réponse. Environ 7 échantillons (et non 10), typiquement entre 2 et 12.

Correction de l'exercice 13 – Calcul d'une somme double

Script.

```
s = 0
for i in range(1, 6):
    for j in range(1, 6):
        s = s + i*j
print(s)
```

— **Factorisation** : la somme se sépare, car $\sum_i \sum_j ij = \left(\sum_i i\right)\left(\sum_j j\right)$ ✓
— **Calcul** :

$$S = 15 \times 15 = 225$$

— **Contrôle partiel** : pour $i = 1$, la somme intérieure vaut $1 + 2 + 3 + 4 + 5 = 15$; pour $i = 2$, elle vaut 30 ; et ainsi de suite ✓
— **Total** : $15 + 30 + 45 + 60 + 75 = 225$ ✓
— **Nombre de tours** : $5 \times 5 = 25$ ✓
— **Version en une boucle** : s = *sum(range(1,6))2* — instantané ✓
— **Généralisation** : $\left(\dfrac{n(n+1)}{2}\right)^2$, soit 225 pour $n = 5$ ✓

— **Coïncidence remarquable** : cette valeur est aussi la **somme des cubes** $1^3 + 2^3 + \cdots + 5^3 = 1 + 8 + 27 + 64 + 125 = 225$ ✓

— **Identité** : $\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k\right)^2$ — une identité classique qu'un script confirme immédiatement ✓

— **Attention** : cette séparation **ne fonctionne pas** pour $\sum_i \sum_j (i+j)$, qui vaut $2n^2 \times \frac{n+1}{2}$ ... plus simplement $n\sum j + n\sum i = 2n \times 15 = 150$ pour $n = 5$ ✓

— **Contrôle** : la moyenne de $i + j$ vaut 6, et $25 \times 6 = 150$ ✓

Réponse. $225 = 15^2$ (la somme double se factorise).

Correction de l'exercice 14 – Recherche dans une liste triée

Fonction.

```
def recherche(L, x):
    a, b = 0, len(L) - 1
    while a <= b:
        m = (a + b) // 2
        if L[m] == x:
            return m
        if L[m] < x:
            a = m + 1
        else:
            b = m - 1
    return -1
```

— **Principe** : on élimine la **moitié** des candidats à chaque étape ✓

— **Coût** : $\log_2 n$ comparaisons ✓

— **Pour** $n = 10^6$: $\log_2 10^6 \approx 19{,}93$, donc **vingt** comparaisons ✓

— **Recherche linéaire** : 10^6 comparaisons dans le pire cas, 5×10^5 en moyenne ✓

— **Gain** : un facteur 50 000 ✓

— **Prérequis absolu** : la liste doit être **triée** — sinon l'algorithme se trompe silencieusement ✓

— **Terminaison** : l'intervalle $[a\,;b]$ **rétrécit** strictement à chaque tour, donc il finit par être vide ✓

— **Invariant** : si x est dans la liste, son indice est dans $[a\,;b]$ ✓

— **Retour** -1 : convention signalant l'absence ✓

— **Division entière** : *(a + b) // 2* est indispensable, un indice devant être **entier** ✓

— **Piège des bornes** : *a = m + 1* et *b = m - 1* (et non *m*) garantissent le rétrécissement — sinon boucle infinie ✓

— **En pratique** : l'opérateur *in* de Python fait une recherche **linéaire** sur une liste ; pour du logarithmique, il faut le module *bisect* ✓

— **Contrôle** : trier d'abord coûte $n \log n$, donc la dichotomie n'est rentable que pour de **nombreuses** recherches sur la même liste ✓

Réponse. 20 comparaisons contre 10^6 — un gain d'un facteur 50 000 (liste triée obligatoire).

Correction de l'exercice 15 – Détecter une boucle infinie

— **a) Oui.** $u_n = 1{,}1^n \to +\infty$, donc 1000 est atteint. Rang : $\frac{\ln 1000}{\ln 1{,}1} \approx 72{,}48$, soit 73 tours ✓

— **b) Oui.** u décroît de 0,0001 par tour, donc $u \to -\infty$. Nombre de tours : $\dfrac{1-0{,}001}{0{,}0001} = 9990$ ✓
— **c) Oui.** Le point fixe est $\dfrac{x}{2} + 0{,}4 = x$, soit $x = 0{,}8 < 1$ — la suite décroît vers 0,8, donc elle passe sous 1 ✓
— **Nombre de tours pour c)** : $u_n = 0{,}8 + 9{,}2 \times 0{,}5^n < 1$ donne $0{,}5^n < 0{,}021739$, soit $n > 5{,}523$ et $n = 6$ tours ✓
— **Vérification** : $u_5 = 0{,}8 + 0{,}2875 = 1{,}0875 > 1$ ✓ et $u_6 = 0{,}8 + 0{,}14375 = 0{,}94375 < 1$ ✓
— **d) Oui, mais pour une mauvaise raison.** Mathématiquement, $\dfrac{1}{2^n}$ n'est **jamais** nul ✓
— **En pratique** : les flottants finissent par **atteindre zéro** par « sous-dépassement » (*underflow*), après environ 1075 divisions ✓
— **Conclusion pour d)** : la boucle se termine, mais grâce à une **limite technique** de la machine, non à un argument mathématique — un code à ne jamais écrire ✓
— **Règle** : ne jamais tester l'égalité stricte d'un flottant à une valeur. Écrire *while u > 1e-15* ✓
— **Contrôle pour d)** : 2^{-1074} est le plus petit flottant positif représentable ✓
— **Piège du cas c)** : si le point fixe valait 1,2 > 1, la suite serait minorée par 1,2 et la boucle **ne s'arrêterait jamais** ✓

Réponse. Les quatre se terminent, mais d) seulement par sous-dépassement des flottants — à éviter.

Correction de l'exercice 16 – Synthèse : étude algorithmique complète

a) **Calculs.**

— $u_1 = 5 + 0{,}3 = 5{,}3$;
— $u_2 = 2{,}65 + 0{,}566038 \approx 3{,}216038$;
— $u_3 \approx 1{,}608019 + 0{,}932825 \approx 2{,}540844$;
— $u_4 \approx 1{,}270418 + 1{,}180714 \approx 2{,}451132$ ✓

b) **Limite.** Le point fixe vérifie $\dfrac{x}{2} + \dfrac{3}{x} = x$, soit $\dfrac{3}{x} = \dfrac{x}{2}$ et $x^2 = 6$.

$$\ell = \sqrt{6} \approx 2{,}4494897$$

— **Justification** : la suite est décroissante (à partir de u_1) et minorée par $\sqrt{6}$, donc elle converge ; sa limite est nécessairement un point fixe positif ✓
— **Minoration** : $u_{n+1} - \sqrt{6} = \dfrac{u_n^2 - 2\sqrt{6}\,u_n + 6}{2u_n} = \dfrac{\left(u_n - \sqrt{6}\right)^2}{2u_n} \geq 0$ ✓
— **Nature** : c'est la **méthode de Newton** appliquée à $x^2 - 6 = 0$ ✓
— **Contrôle** : $u_4 \approx 2{,}451132$, déjà à $1{,}6 \times 10^{-3}$ de $\sqrt{6}$ ✓

c) **Script.**

```
u = 10.0
n = 0
while abs(u*u - 6) >= 1e-8:
    u = u/2 + 3/u
    n = n + 1
print(n, u)
```

— **Écarts** $\left|u_n^2 - 6\right|$: 94 ; 22,09 ; 4,3429 ; 0,45590 ; 0,0080478 ; $2{,}695 \times 10^{-6}$; $3{,}038 \times 10^{-13}$ ✓

— **Réponse** : le script effectue **six** tours et affiche $u \approx 2{,}4494897$ ✓

— **Convergence quadratique** : l'écart passe de 0,008 à $2{,}7 \times 10^{-6}$ puis à $3{,}0 \times 10^{-13}$ — le nombre de décimales **double** à chaque tour ✓

— **Terminaison** : garantie par cette convergence, pourvu que $u_0 > 0$ ✓

— **Comparaison avec la dichotomie** : pour la même précision sur $[2\,;3]$, il faudrait environ 27 étapes ✓

— **Contrôle** : $\log_2\left(\dfrac{1}{10^{-8}}\right) \approx 26{,}6$ ✓

— **Danger** : avec $u_0 = 0$, division par zéro ; avec $u_0 < 0$, la suite converge vers $-\sqrt{6}$, et le critère serait tout de même satisfait ✓

— **Généralisation** : $u_{n+1} = \dfrac{u_n}{2} + \dfrac{a}{2u_n}$ converge vers $\sqrt{2a}$… ici $\dfrac{u_n}{2} + \dfrac{3}{u_n}$ correspond à $a = 3$ dans la forme $\dfrac{u_n}{2} + \dfrac{a}{u_n}$, dont la limite est $\sqrt{2a} = \sqrt{6}$ ✓

Réponse. a) 5,3 ; 3,216 ; 2,541 ; 2,451 · b) limite $\sqrt{6}$ (méthode de Newton) · c) six tours.