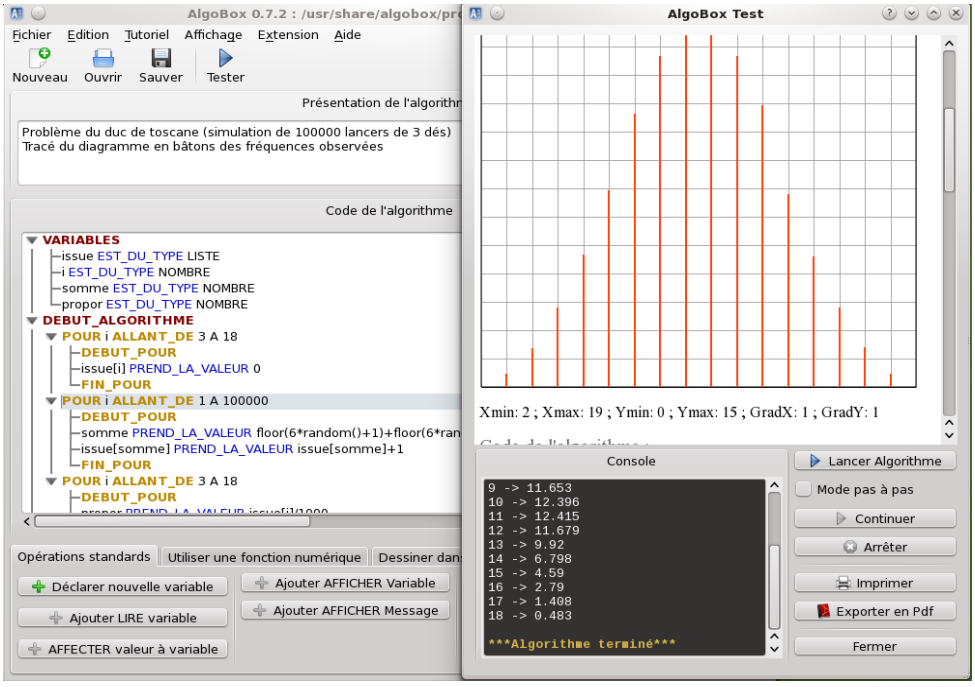
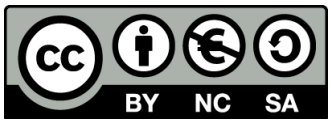
Algorithmique en classe de terminale avec AlgoBox

(programme obligatoire)



Version 1.0 - Mai 2013



Cette brochure a été réalisée avec le système de composition LaTeX et l'éditeur TeXMAKER .
http://www.xm1math.net/doculatex/index.html

Sommaire

Avant-propos

Rappel des instructions officielles concernant l'algorithmique dans les programmes de mathématiques :

1. ***Instructions élémentaires (affectation, calcul, entrée, sortie).***
   Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables :
   – *d'écrire une formule permettant un calcul ;*
   – *d'écrire un programme calculant et donnant la valeur d'une fonction ;*
   – *ainsi que les instructions d'entrées et sorties nécessaires au traitement.*
2. ***Boucle et itérateur, instruction conditionnelle.***
   Les élèves, dans le cadre d'une résolution de problèmes, doivent être capables de :
   – *programmer un calcul itératif, le nombre d'itérations étant donné ;*
   – *programmer une instruction conditionnelle, un calcul itératif, avec une fin de boucle conditionnelle.*
3. ***Dans le cadre de cette activité algorithmique, les élèves sont entraînés à :***
   – *décrire certains algorithmes en langage naturel ou dans un langage symbolique ;*
   – *en réaliser quelques-uns à l'aide d'un tableur ou d'un programme sur calculatrice ou avec un logiciel adapté ;*
   – *interpréter des algorithmes plus complexes.*

Contenu de cette brochure :

– Des activités « élèves » strictement conformes aux programmes en vigueur.
– Des annexes comportant :
  – Des activités d'apprentissage des techniques de base en algorithmique avec Algobox ;
  – Un mémento sur les fonctions d'AlgoBox ;
  – Des algorithmes supplémentaires en rapport avec le contenu mathématique des programmes de première.

À propos des activités « élèves » :
Les fiches « professeurs » et « élèves » sont sur des pages différentes afin de faciliter les photocopies.
Les activités sont présentées ici sous forme d'énoncés « à trou ». Il est bien sur possible de les adapter selon sa convenance.
Adaptations possibles :
– donner l'algorithme complet et demander de décrire ce qu'il fait ;
– demander la réalisation complète de l'algorithme à partir de zéro.

Les fichiers AlgoBox des algorithmes de la partie « Activités élèves » et de l'annexe C sont disponibles en ligne à l'adresse suivante : `http://www.xm1math.net/algobox/algobook.html`

Première partie

Activités « élèves »

1 Fonctions

Fiche professeur 1A

- Fiche élève correspondante : page 4
- *Fichier AlgoBox associé (algorithme complet) :* `algo_1A.alg`
- *Contexte (TS/TES) :* Recherche par dichotomie d'une valeur approchée d'un antécédent avec une fonction décroissante .
- *Prolongements possibles :*
  - Remplacer la boucle `POUR numero_etape ALLANT_DE 1 A 4` par un `TANT_QUE` portant sur la précision souhaitée.
  - Étudier un autre cas où la fonction est strictement croissante (une activité correspondante est disponible dans la brochure de première)
  - Établir un algorithme qui fonctionne dans tous les cas (fonction strictement croissante ou strictement décroissante)

Fiche professeur 1B

- Fiche élève correspondante : page 6
- *Fichier AlgoBox associé (algorithme complet) :* `algo_1B.alg`
- *Contexte (TS/TSTI2D/TSTL) :* Recherche d'un seuil entier à partir duquel la valeur d'une fonction devient inférieure à 1.
- *Prolongement possible :*
  - Justifier que la fonction est bien décroissante sur $[10 ; +\infty[$.

Fiche professeur 1C

- Fiche élève correspondante : page 7
- *Fichier AlgoBox associé (algorithme complet) :* `algo_1C.alg`
- *Contexte (TS) :* Utilisation de la méthode d'Euler pour construire une approximation de la courbe de la fonction f telle que $f(0) = 1$ et $f'(x) = f(x)$ avec un pas de 0,1.
- *Prolongement possible :* Faire tracer les points obtenus dans un repère à l'aide de l'onglet « Dessiner dans un repère ».

Fiche professeur 1D

- Fiche élève correspondante : page 9
- *Fichier AlgoBox associé (algorithme complet) :* `algo_1D.alg`
- *Contexte (TS/TES/TSTI2D/TSTL) :* Recherche d'un seuil entier à partir duquel la valeur d'une fonction logarithmique devient supérieure à 120.

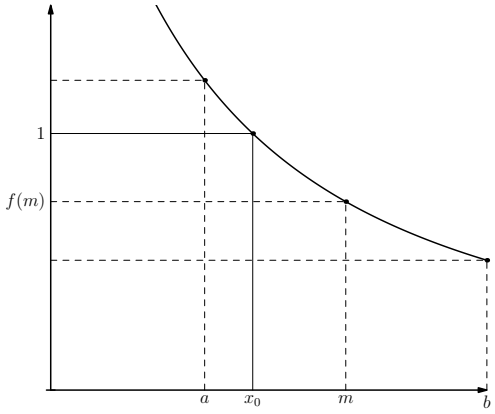
Fiche professeur 1E

– Fiche élève correspondante : page 10
– *Fichiers AlgoBox associés (algorithmes complets) :* `algo_1E.alg` et `algo_1Ebis.alg`
– *Contexte (TS/TES/TSTI2D/TSTL) :* Détermination d'une valeur approchée de l'aire sous une courbe à l'aide de rectangles (la méthode présentée ici ne donne qu'un minorant de l'aire)
– *Prolongement possible :* Détermination d'un majorant de l'aire en changeant la définition des rectangles.

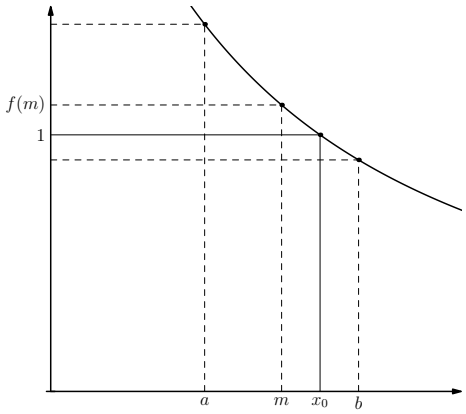
Fiche élève 1A

Soit f la fonction définie sur $[1\,;2]$ par $f(x)=\dfrac{2\sqrt{x}}{x^2}$.

1. Dériver f et montrer que, pour tout $1 \leqslant x \leqslant 2$, $f'(x)$ peut s'écrire sous la forme $f'(x)=\dfrac{-3}{x^2\sqrt{x}}$.
2. Justifier que l'équation $f(x)=1$ admet une unique solution x_0 dans $[1\,;2]$.
3. Pour déterminer une valeur approchée de x_0, on utilise la méthode dite de la « dichotomie » dont le principe consiste à couper l'intervalle en deux et à regarder de quel côté se situe la solution par rapport au milieu de l'intervalle.
   a) Étant donné un intervalle $[a;b]$ de milieu m et contenant x_0 (avec $a \geqslant 1$ et $b \leqslant 2$).
   - Si $f(m)<1$, dans quel intervalle se situe x_0 ?



   - Si $f(m)>1$, dans quel intervalle se situe x_0 ?



   b) Compléter le tableau suivant :

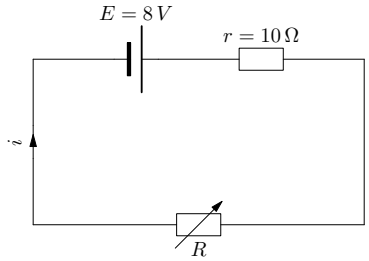
Étape	Intervalle de départ $[a;b]$	milieu m	$f(m)<1$?	Nouvel intervalle $[a;b]$
1	$a=1$; $b=2$	$m=1.5$	NON	$a=$; $b=$
2	$a=$; $b=$	$m=$		$a=$; $b=$
3	$a=$; $b=$	$m=$		$a=$; $b=$
4	$a=$; $b=$	$m=$		$a=$; $b=$

c) On cherche à automatiser les calculs grâce à un algorithme. Compléter les lignes 14 et 18 pour que l'algorithme AlgoBox ci-dessous réponde au problème.

```
 1: VARIABLES
 2: a EST_DU_TYPE NOMBRE
 3: b EST_DU_TYPE NOMBRE
 4: m EST_DU_TYPE NOMBRE
 5: numero_etape EST_DU_TYPE NOMBRE
 6: DEBUT_ALGORITHME
 7:     a PREND_LA_VALEUR 1
 8:     b PREND_LA_VALEUR 2
 9:     POUR numero_etape ALLANT_DE 1 A 4
10:         DEBUT_POUR
11:         m PREND_LA_VALEUR (a+b)/2
12:         SI (2*sqrt(m)/(m*m)<1) ALORS
13:             DEBUT_SI
14:             ..... PREND_LA_VALEUR m
15:             FIN_SI
16:         SINON
17:             DEBUT_SINON
18:             ..... PREND_LA_VALEUR m
19:             FIN_SINON
20:         AFFICHER a
21:         AFFICHER " <x0< "
22:         AFFICHER b
23:         FIN_POUR
24: FIN_ALGORITHME
```

Fiche élève 1B

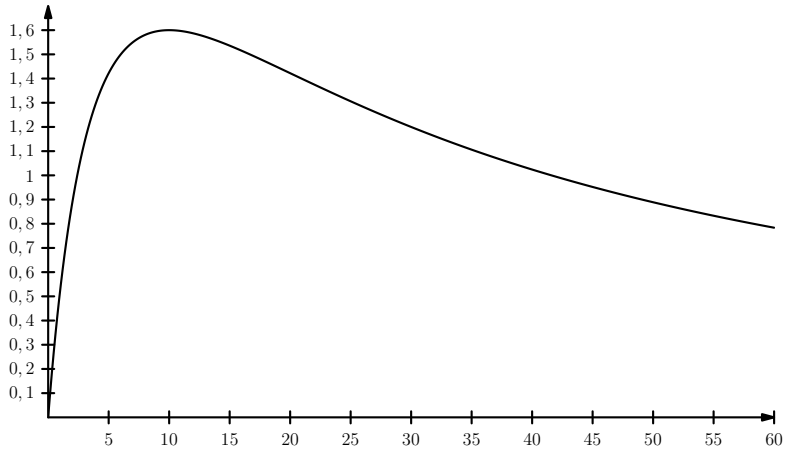
On considère le circuit ci-dessous dans lequel R est variable.



La puissance dissipée (en watts) dans le résistor de résistance R est :

$$P = \frac{E^2 \times R}{(R+r)^2} = \frac{64 \times R}{(R+10)^2}$$

La courbe représentant P (en ordonnée) en fonction de R en (abscisse) est donnée ci-dessous :



1. Déterminer, par le calcul, la valeur de P quand $R = 10\,\Omega$.
2. Le graphique permet-il d'établir une conjecture viable sur la valeur vers laquelle tend P quand R devient très grand ?
3. Déterminer la limite de P quand R tend vers $+\infty$.
4. On cherche à déterminer, à l'aide d'un algorithme, la première valeur entière de R ($R \geqslant 10$) pour laquelle la puissance P devient inférieure à 1 W.
   Pour cela, on part de $R = 10$ et on augmente R de 1 tant que cela est nécessaire.
   Compléter les lignes 6 et 7 de l'algorithme AlgoBox ci-dessous pour qu'il réponde au problème.

```
 1: VARIABLES
 2: R EST_DU_TYPE NOMBRE
 3: P EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5: | R PREND_LA_VALEUR 10
 6: | P PREND_LA_VALEUR .......
 7: | TANT_QUE (P........) FAIRE
 8: | | DEBUT_TANT_QUE
 9: | | R PREND_LA_VALEUR R+1
10: | | P PREND_LA_VALEUR 64*R/pow(R+10,2)
11: | | FIN_TANT_QUE
12: | AFFICHER R
13: FIN_ALGORITHME
```

Fiche élève 1C

Approximation de la courbe sur $[0;1]$ de la fonction f telle que $f(0)=1$ et $f'(x)=f(x)$ pour tout x.

► **Principe général :** Soit f une fonction dérivable sur un intervalle I contenant a.
Pour tout x de I « proche de a » , on a $f(x)\approx f(a)+f'(a)(x-a)$.
Donc en connaissant $f(a)$ et $f'(a)$, on peut en déduire une approximation de $f(x)$. De proche en proche, on peut donc en déduire un tableau de valeurs approchées de $f(x)$ qui permet de construire une approximation de la courbe de f.

1. On a $f(0)=1$ donc le premier point de la courbe a pour coordonnées $\begin{cases} x_1=0 \\ y_1=1 \end{cases}$.

   – Pour x « proche de 0 » , $f(x)\approx f(0)+f'(0)\times(x-0)$.

   Donc, $f(0{,}1)\approx f(0)+f'(0)\times(0{,}1-0)\Leftrightarrow f(0{,}1)\approx f(0)+f'(0)\times\ldots\ldots$

   Or, $f'(0)=f(0)$. On en déduit que $f(0{,}1)\approx f(0)+f(0)\times 0{,}1\Leftrightarrow f(0{,}1)\approx f(0)\times\ldots\ldots\approx$.

   Le deuxième point de la courbe a pour coordonnées $\begin{cases} x_2=0{,}1 \\ y_2= \end{cases}$. Placer ce point dans le graphique.

   – Pour x « proche de 0,1 » , $f(x)\approx f(0{,}1)+f'(0{,}1)\times(x-0{,}1)$.

   Donc, $f(0{,}2)\approx f(0{,}1)+f'(0{,}1)\times(0{,}2-0{,}1)\Leftrightarrow f(0{,}2)\approx f(0{,}1)+f'(0{,}1)\times\ldots\ldots$

   Or, $f'(0{,}1)=f(0{,}1)$. On en déduit que $f(0{,}2)\approx f(0{,}1)+f(0{,}1)\times 0{,}1\Leftrightarrow f(0{,}2)\approx f(0{,}1)\times\ldots\ldots\approx$.

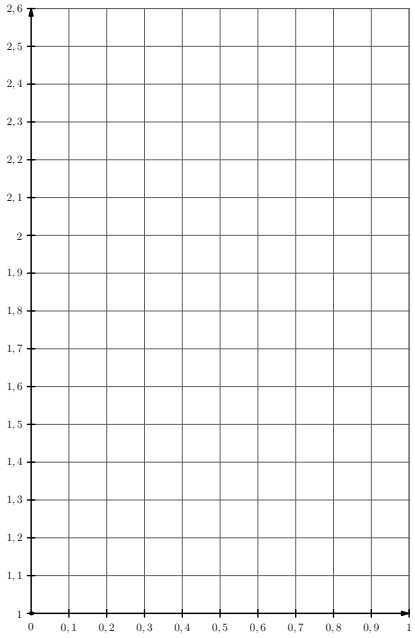
   Le troisième point de la courbe a pour coordonnées $\begin{cases} x_3=0{,}2 \\ y_3= \end{cases}$. Placer ce point dans le graphique.

   – Pour x « proche de 0,2 » , $f(x)\approx f(0{,}2)+f'(0{,}2)\times(x-0{,}2)$.

   Donc, $f(0{,}3)\approx f(0{,}2)+f'(0{,}2)\times(0{,}2-0{,}1)\Leftrightarrow f(0{,}3)\approx f(0{,}2)+f'(0{,}2)\times\ldots\ldots$

   Or, $f'(0{,}2)=f(0{,}2)$. On en déduit que $f(0{,}3)\approx f(0{,}2)+f(0{,}2)\times 0{,}1\Leftrightarrow f(0{,}3)\approx f(0{,}2)\times\ldots\ldots\approx$.

   Le quatrième point de la courbe a pour coordonnées $\begin{cases} x_4=0{,}3 \\ y_4= \end{cases}$. Placer ce point dans le graphique.



2. On cherche à créer un algorithme qui permette d'automatiser les calculs.
   a) Comment passe-t-on de l'abscisse d'un point à celle du point suivant ?

b) Comment passe-t-on de y_1 à y_2 ?

c) De la même façon, exprimer y_3 en fonction de y_2 : $y_3 =$

d) En déduire comment calculer l'ordonnée d'un nouveau point à partir des coordonnées (X,Y) du point précédent :

Le nouvel Y =

e) Compléter les lignes 12 et 13 de l'algorithme AlgoBox ci-dessous pour qu'il réponde au problème :

```
 1: VARIABLES
 2: X EST_DU_TYPE NOMBRE
 3: Y EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:     X PREND_LA_VALEUR 0
 6:     Y PREND_LA_VALEUR 1
 7:     TANT_QUE (X<1) FAIRE
 8:         DEBUT_TANT_QUE
 9:         AFFICHER X
10:         AFFICHER " -> "
11:         AFFICHER Y
12:         Y PREND_LA_VALEUR ......................
13:         X PREND_LA_VALEUR ......................
14:         FIN_TANT_QUE
15: FIN_ALGORITHME
```

3. En exécutant l'algorithme, compléter le tableau ci-dessous donnant les coordonnées des points de la courbe :

X	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
$Y \approx$	1										

Compléter le graphique à l'aide de ces coordonnées.

4. Quelle est la nature de la suite $(y_n)_{n\geqslant 1}$?

Fiche élève 1D

Quand l'oreille d'un individu est soumise à une pression acoustique x, exprimée en bars, l'intensité sonore, exprimée en décibels, du bruit responsable de cette pression est donnée par :

$$f(x) = 8{,}68 \times \ln x + 93{,}28$$

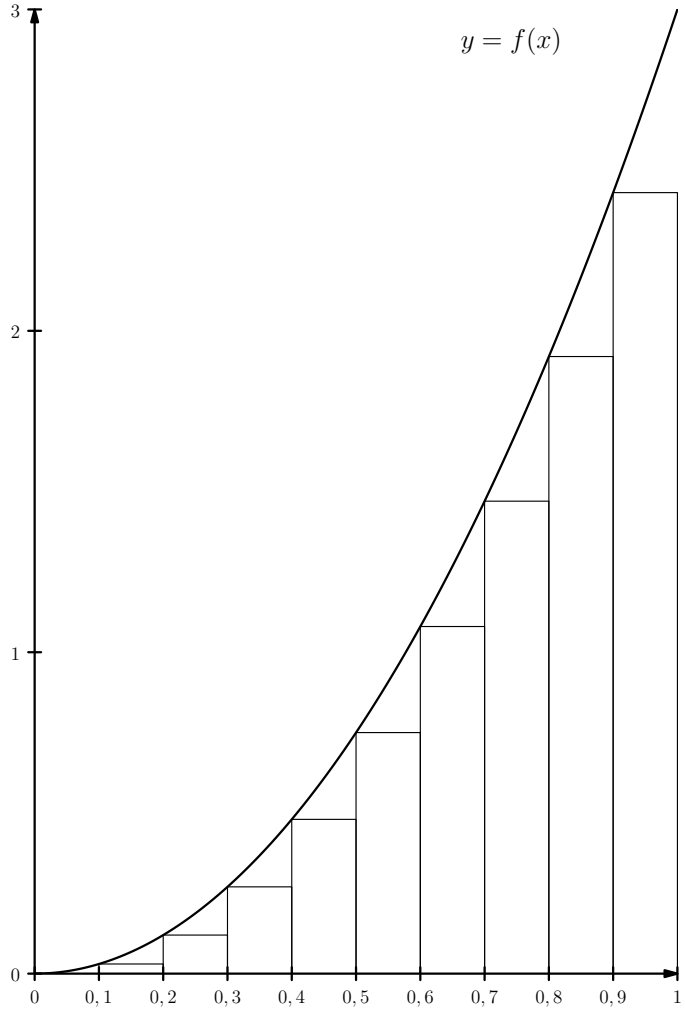
1. Calculer l'intensité sonore correspondante à une pression acoustique de 5 bars.
2. Justifier que f est une fonction strictement croissante sur $]0\,;+\infty[$.
3. Déterminer la limite de f en $+\infty$.
4. Un individu normal ne peut supporter un bruit supérieur à 120 décibels. On cherche à connaitre le premier nombre entier x de bars pour lequel l'intensité $f(x)$ dépasse 120 décibels à l'aide d'un algorithme.
   Pour cela on part d'une pression $x = 1$ que l'on augmente de 1 tant que cela est nécessaire.
   Compléter les lignes 5 et 7 de l'algorithme AlgoBox ci-dessous pour qu'il réponde au problème :
   (attention : la syntaxe informatique pour le logarithme népérien est `log` *et pas* `ln`)

```
1:  VARIABLES
2:  x EST_DU_TYPE NOMBRE
3:  DEBUT_ALGORITHME
4:  |  x PREND_LA_VALEUR 1
5:  |  TANT_QUE (8.68*log(x)+93.28.....) FAIRE
6:  |  |  DEBUT_TANT_QUE
7:  |  |  x PREND_LA_VALEUR .....
8:  |  |  FIN_TANT_QUE
9:  |  AFFICHER x
10: FIN_ALGORITHME
```

Fiche élève 1E

Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = 3x^2$. On cherche à déterminer une valeur approchée de l'aire sous la courbe sur $[0\,;1]$ en utilisant des rectangles.

1. On découpe l'intervalle $[0\,;1]$ en 10 intervalles et on construit des rectangles de la façon suivante :



La somme des aires des rectangles permet de déterminer une valeur approchée de l'aire sous la courbe de f sur $[0\,;1]$.

a) Quelle est la largeur de chaque rectangle ? ☐

b) Quelle est la hauteur du premier rectangle ? ☐ Quelle est son aire ? ☐

c) Quelle est la hauteur du deuxième rectangle ? ☐ Quelle est son aire ? ☐

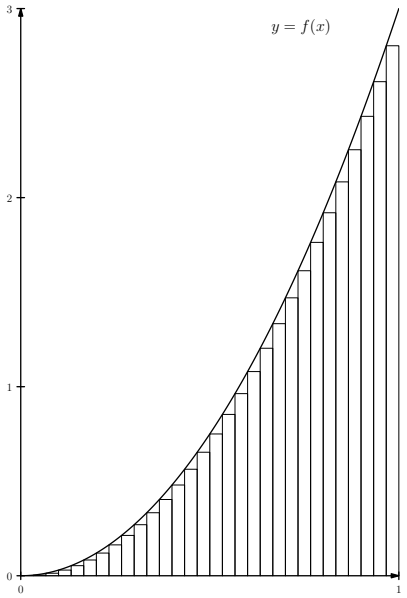
d) On cherche à effectuer la somme des aires des rectangles avec un algorithme en se basant sur le principe suivant :
On utilise une variable `aire` *qui sert à stocker la somme des aires des rectangles au fur et à mesure. On part de* $x = 0.1$ *et on ajoute à la variable* `aire` *l'aire du rectangle commençant à* x*, puis on continue le processus en augmentant* x *de* 0.1 *tant que c'est nécessaire.*

Compléter les lignes 7 et 9 de l'algorithme AlgoBox ci-dessous pour qu'il réponde au problème :

```
1:  VARIABLES
2:  x EST_DU_TYPE NOMBRE
3:  aire EST_DU_TYPE NOMBRE
4:  DEBUT_ALGORITHME
5:      aire PREND_LA_VALEUR 0
6:      x PREND_LA_VALEUR 0.1
7:      TANT_QUE (x<=........) FAIRE
8:          DEBUT_TANT_QUE
9:          aire PREND_LA_VALEUR aire+............
10:         x PREND_LA_VALEUR x+0.1
11:         FIN_TANT_QUE
12:     AFFICHER aire
13: FIN_ALGORITHME
```

e) Quel est le résultat affiché lors de l'exécution de l'algorithme ?

2. Une augmentation du nombre de rectangles doit permettre d'obtenir une meilleure approximation :



On cherche à adapter l'algorithme précédent en se basant cette fois-ci sur une découpage de l'intervalle $[0\,;1]$ en 1000 intervalles (de 0 à 0.001, de 0.001 à 0.002, etc.).

a) Compléter les lignes 6, 7, 9 et 10 de l'algorithme AlgoBox ci-dessous pour qu'il réponde au problème :

```
1:  VARIABLES
2:  x EST_DU_TYPE NOMBRE
3:  aire EST_DU_TYPE NOMBRE
4:  DEBUT_ALGORITHME
5:      aire PREND_LA_VALEUR 0
6:      x PREND_LA_VALEUR ..........
7:      TANT_QUE (x<=........) FAIRE
8:          DEBUT_TANT_QUE
9:          aire PREND_LA_VALEUR aire+............
10:         x PREND_LA_VALEUR x+...........
11:         FIN_TANT_QUE
12:     AFFICHER aire
13: FIN_ALGORITHME
```

b) Quel est le résultat affiché lors de l'exécution de l'algorithme ?

2 Suites

Fiche professeur 2A

– Fiche élève correspondante : page 13
– *Fichier AlgoBox associé (algorithme complet) :* `algo_2A.alg`
– *Contexte (TS/TES/TSTI2D/TSTL) :* Détermination du rang à partir duquel les termes d'une suite deviennent inférieurs à 1.

Fiche professeur 2B

– Fiche élève correspondante : page 14
– *Fichiers AlgoBox associés (algorithme complet) :* `algo_2B.alg` et `algo_2Bbis.alg`
– *Contexte (TS) :* Calcul des termes d'une suite définie comme une somme et détermination du rang à partir duquel les termes de la suite deviennent supérieurs à 5.

Fiche professeur 2C

– Fiche élève correspondante : page 15
– *Fichier AlgoBox associé (algorithme complet) :* `algo_2C.alg`
– *Contexte (TS) :* Estimation de la « rapidité de convergence » d'une suite après étude de la suite en question.

Fiche professeur 2D

– Fiche élève correspondante : page 17
– *Fichier AlgoBox associé (algorithme complet) :* `algo_2D.alg`
– *Contexte (TS/TES) :* Calcul des termes d'une suite arithmético-géométrique.
– *Prolongement possible :* Détermination du rang à partir duquel la suite franchit un certain seuil.

Fiche élève 2A

Une plaque de verre teintée est telle qu'un rayon lumineux qui la traverse perd 20 % de son intensité lumineuse et on fait traverser à un rayon lumineux d'intensité 50 cd une série de ces plaques de verre teintée.
On note $I_0 = 50$ et I_n l'intensité du rayon lumineux après le passage de n plaques.

1. Justifier que la suite (I_n) est géométrique et donner sa raison.
2. Exprimer I_n en fonction de n.
3. Calculer l'intensité du rayon lumineux après le passage de 4 plaques.
4. Justifier que la suite (I_n) est décroissante.
5. On cherche à déterminer le plus petit nombre de plaques que le rayon lumineux doit franchir pour que son intensité devienne inférieure à 1 cd.
   a) **Méthode 1 : avec un algorithme**.
   Compléter les lignes 7 et 9 de l'algorithme AlgoBox ci-dessous pour qu'il réponde à la question.

```
1:  VARIABLES
2:  n EST_DU_TYPE NOMBRE
3:  I EST_DU_TYPE NOMBRE
4:  DEBUT_ALGORITHME
5:      n PREND_LA_VALEUR 0
6:      I PREND_LA_VALEUR 50
7:      TANT_QUE (.........) FAIRE
8:          DEBUT_TANT_QUE
9:          I PREND_LA_VALEUR I*...........
10:         n PREND_LA_VALEUR n+1
11:         FIN_TANT_QUE
12:     AFFICHER n
13: FIN_ALGORITHME
```

   b) **Méthode 2 : en résolvant une inéquation**.
   Déterminer, par le calcul, le plus petit entier n tel que $I_n \leqslant 1$.

Fiche élève 2B

Soit (U_n) la suite définie par $U_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ (pour $n \geqslant 1$).

1. Donner les valeurs de U_1, U_2 et U_3.
2. Compléter la ligne 8 de l'algorithme AlgoBox ci-dessous pour qu'il affiche la valeur de U_{20}.

```
 1: VARIABLES
 2: U EST_DU_TYPE NOMBRE
 3: n EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   U PREND_LA_VALEUR 1
 6:   POUR n ALLANT_DE 2 A 20
 7:     DEBUT_POUR
 8:     U PREND_LA_VALEUR U+........
 9:     FIN_POUR
10:   AFFICHER U
11: FIN_ALGORITHME
```

3. Exprimer $U_{n+1} - U_n$ en fonction de n. Que peut-on en déduire sur le sens de variation de la suite (U_n) ?
4. Étudier les variations sur $[0\,;+\infty[$ de la fonction f définie par $f(x) = e^x - x - 1$. En déduire que, pour tout entier $n \geqslant 1$, $e^{\frac{1}{n}} \geqslant \dfrac{n+1}{n}$.
5. Montrer que, pour tout entier $n \geqslant 1$, $e^{U_n} \geqslant n + 1$.
6. En déduire la limite de (U_n).
7. On cherche à déterminer, à l'aide d'un algorithme, le plus petit entier n à partir duquel $U_n \geqslant 5$.
   Compléter les lignes 7 et 10 de l'algorithme AlgoBox ci-dessous pour qu'il réponde à la question.

```
 1: VARIABLES
 2: U EST_DU_TYPE NOMBRE
 3: n EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   U PREND_LA_VALEUR 1
 6:   n PREND_LA_VALEUR 1
 7:   TANT_QUE (U..........) FAIRE
 8:     DEBUT_TANT_QUE
 9:     n PREND_LA_VALEUR n+1
10:     U PREND_LA_VALEUR U+.......
11:     FIN_TANT_QUE
12:   AFFICHER n
13: FIN_ALGORITHME
```

Fiche élève 2C

On considère :
– f la fonction définie sur $\mathbb{R}$ par $f(x) = 1 + \mathrm{e}^{x-2}$;
– (U_n) la suite définie par $U_0 = 0$ et $U_{n+1} = f(U_n) = 1 + \mathrm{e}^{U_n - 2}$.

Partie A : *Étude de la convergence de la suite* (U_n)

1. Montrer par récurrence que, pour tout entier positif n, on a $0 \leqslant U_n \leqslant 2$.
   (On encadrera d'abord $U_p - 2$*, puis* $e^{U_p - 2}$ *et enfin* $1 + e^{U_p - 2}$*)*
2. Montrer par récurrence que, pour tout entier positif n, on a $U_{n+1} \geqslant U_n$.
3. À l'aide d'un théorème du cours, déduire des deux questions précédente que la suite (U_n) converge vers une limite que l'on notera l.
4. Justifier que l'on doit avoir $l = 1 + \mathrm{e}^{l-2}$, avec $0 \leqslant l \leqslant 2$, en complétant les phrases ci-dessous :

   Comme $\lim\limits_{n \to +\infty} U_n = l$ et que f est $\boxed{}$ sur $[0\,;2]$,

   on a $\lim\limits_{n \to +\infty} f(U_n) = \boxed{}$. Donc $\lim\limits_{n \to +\infty} U_{n+1} = \boxed{}$.

   On en déduit que $l = \boxed{}$,c'est à dire que $l = 1 + \mathrm{e}^{l-2}$.

Partie B : *Recherche de la valeur de l*
Soit g la fonction définie sur $\mathbb{R}$ par $g(x) = f(x) - x = 1 + \mathrm{e}^{x-2} - x$.

1. Justifier que dire que l vérifie la relation $l = 1 + \mathrm{e}^{l-2}$ équivaut à dire que $g(l) = 0$.
2. En étudiant le signe de $g'(x)$, justifier les variations de g figurant dans le tableau ci-dessous :
   (on se s'intéresse pas aux limites de g)

x	$-\infty$		2		$+\infty$
$g(x)$		↘	0	↗	

3. On constate, d'après le tableau de variations ci-dessus, que g ne peut s'annuler qu'une seule fois sur l'intervalle $[0\,;2]$. En déduire la valeur de l.

Partie C : *Étude expérimentale et théorique de la rapidité de convergence de la suite* (U_n)

On considère la suite (V_n) définie par $V_n = \dfrac{U_{n+1} - 2}{U_n - 2}$ et l'algorithme AlgoBox ci-dessous :

```
 1: VARIABLES
 2: i EST_DU_TYPE NOMBRE
 3: U EST_DU_TYPE NOMBRE
 4: ancienU EST_DU_TYPE NOMBRE
 5: V EST_DU_TYPE NOMBRE
 6: DEBUT_ALGORITHME
 7:    U PREND_LA_VALEUR 0
 8:    POUR i ALLANT_DE 1 A .....
 9:    DEBUT_POUR
10:    ancienU PREND_LA_VALEUR U
11:    U PREND_LA_VALEUR 1+exp(U-2)
12:    V PREND_LA_VALEUR (U-2)/(ancienU-2)
13:    AFFICHER V
14:    FIN_POUR
15: FIN_ALGORITHME
```

Fonctionnement de l'algorithme : lorsque `i` vaut 1 ;
– après exécution de la ligne 10, `ancienU` contient la valeur de U_0.
– après exécution de la ligne 11, `U` contient la valeur de U_1.
– après exécution de la ligne 12, `V` contient la valeur de $\dfrac{U_1 - 2}{U_0 - 2} = V_0$.

1. Compléter les phrases ci-dessous : lorsque `i` vaut 2 ;
   – après exécution de la ligne 10, `ancienU` contient la valeur de $U_{\dots}$
   – après exécution de la ligne 11, `U` contient la valeur de $U_{\dots}$
   – après exécution de la ligne 12, `V` contient la valeur de $\dfrac{U_{\dots}-2}{U_{\dots}-2} = V_{\dots}$.
2. Indiquer ci-dessous la valeur qu'il faut insérer à la fin de la ligne 8 pour que l'algorithme affiche les valeurs des termes de la suite (V_n) jusqu'à V_{100} ?

   Réponse : ☐
3. En exécutant l'algorithme (après l'avoir complété), indiquer ci-dessous la dernière valeur affichée (correspondante à V_{100}) :

   Dernière valeur affichée : ☐
4. Vers quel nombre semblent tendre les termes de la suite (V_n) ?

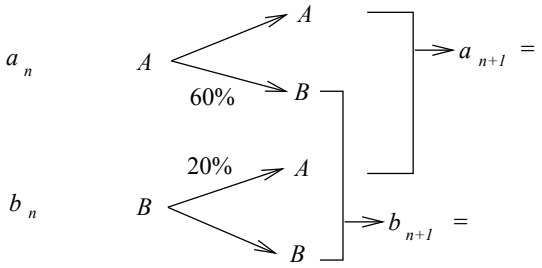
   Réponse : ☐

   (on parle alors de « convergence lente » pour la suite (U_n)).

Fiche élève 2D

Un guide classe les campings selon deux catégories A et B. Chaque année 60 % des campings de la catégorie A passent dans la catégorie B et 20 % des campings de la catégorie B passent dans la catégorie A.
On note a_n, la proportion de campings classés dans la catégorie A et b_n, la proportion de campings classés dans la catégorie B en l'année $(2010+n)$.
On suppose qu'en 2010, 3/4 des campings sont classés dans la catégorie A. On a donc $a_0 = 0{,}75$ et $b_0 = 0{,}25$.

1. Déterminer a_{n+1} et b_{n+1} en fonction de a_n et b_n.



2. En utilisant que, pour tout entier n, $a_n + b_n = 1$, montrer que $a_{n+1} = 0{,}2\,a_n + 0{,}2$.
3. On cherche à déterminer à l'aide d'un algorithme la valeur de a_{10}.
   Compléter les lignes 5 et 6 de l'algorithme AlgoBox ci-dessous pour qu'il réponde à la question.

```
1:  VARIABLES
2:  a EST_DU_TYPE NOMBRE
3:  n EST_DU_TYPE NOMBRE
4:  DEBUT_ALGORITHME
5:  | a PREND_LA_VALEUR ........
6:  | POUR n ALLANT_DE 1 A ......
7:  | | DEBUT_POUR
8:  | | a PREND_LA_VALEUR 0.2*a+0.2
9:  | | FIN_POUR
10: | AFFICHER a
11: FIN_ALGORITHME
```

4. Montrer que la suite (c_n) définie par $c_n = a_n - 0{,}25$ est une suite géométrique dont on donnera le premier terme c_0 et la raison.
5. Déterminer c_n en fonction de n.
6. En déduire a_n en fonction de n.
7. Déterminer le sens de variation de la suite (a_n).
8. Déterminer la limite de la suite (a_n).

3 Probabilités

Fiche professeur 3A

- Fiche élève correspondante : page 19
- *Fichier AlgoBox associé (algorithme complet) :* `algo_3A.alg`
- *Contexte (TS/TES) :* Algorithme de simulation lié à des probabilités conditionnelles.
- *Prolongements possibles :*
  - Faire calculer et afficher la fréquence de numéros 2 et 3 tirés pour le cadre C.
  - Faire afficher l'évolution des fréquences tous les 1000 tirages.

Fiche professeur 3B

- Fiche élève correspondante : page 21
- *Fichier AlgoBox associé (algorithme complet) :* `algo_3B.alg`
- *Contexte (TS/TES/TSTI2D/TSTL) :* Algorithme de simulation d'une marche aléatoire - Réinvestissement de loi binomiale vu en première.
- *Prolongement possible :* Ajouter un algorithme simulant une seule partie mais dans lequel on montrerait graphiquement les mouvements du jeton dans le repère d'AlgoBox.

Fiche professeur 3C

- Fiche élève correspondante : page 23
- *Fichier AlgoBox associé (algorithme complet) :* `algo_3C.alg`
- *Contexte (TS/TES/TSTI2D/TSTL) :* Algorithme de simulation d'une variable aléatoire égale au maximum de deux variables aléatoires suivant une loi uniforme.
- *Prolongements possibles :*
  - Diminuer le nombre de simulations et faire dessiner les points de coordonnées X et Y tels que le max soit inférieur à 5 dans le repère d'AlgoBox. Faire le lien ensuite avec l'aspect géométrique du problème.
  - Proposer le même genre d'activité en remplaçant le maximum par le minimum.

Fiche élève 3A

Trois cadres d'une société fabriquant des jeux de sociétés (représentés par les lettres A, B et C) se disputent trois places de parking (numérotées 1,2 et 3 - la place numéro 1 étant mieux située que la place numéro 2 ; la place numéro 3 étant la plus mal située). Pour départager les cadres, le directeur des ressources humaines propose d'utiliser un dé spécial (fabriqué dans l'usine) comportant deux faces « 1 » , deux faces « 2 » et deux faces « 3 » (ce qui revient à choisir au hasard de façon équiprobable un entier compris entre 1 et 3) en suivant la démarche ci-dessous :

– Le cadre A lance le premier le dé lui indiquant le numéro de la place de parking qui lui sera attribué ;
– Le cadre B lance à son tour le dé. S'il tombe sur le numéro de la place déjà attribuée au cadre A, il doit relancer le dé tant qu'il n'obtient pas un numéro différent de celui du cadre A.
– Le cadre C se voit alors attribué le numéro de la place de parking non encore attribuée.

Le cadre C s'oppose à la démarche proposée par le directeur des ressources humaines car il s'estime désavantagé par le fait qu'il n'y ait plus de choix quand arrive son tour.
Qu'en-est-il vraiment ?

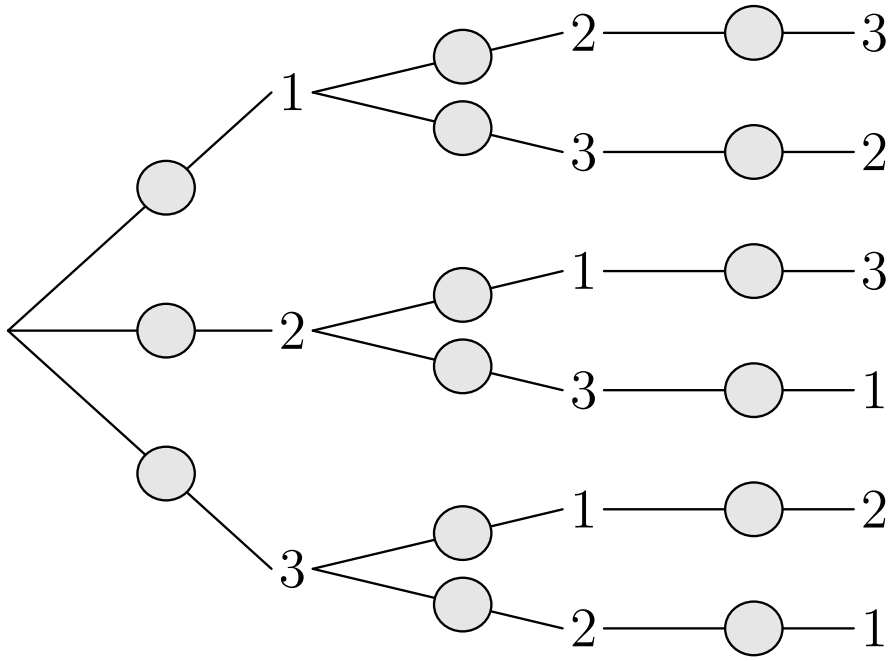
Partie A : *Simulation avec un algorithme*

1. Quelque soit la situation, que vaut toujours la somme des numéros attribués aux trois cadres ?
2. À l'aide de l'algorithme AlgoBox ci-dessous, on cherche à déterminer expérimentalement une estimation de la probabilité que le cadre C obtienne la place de parking numéro 1 en simulant 100 000 fois la démarche proposée par le directeur des ressources humaines.
   Remarques :
   – *la fonction `ALGOBOX_ALEA_ENT(1,3)` permet d'obtenir un entier pseudo-aléatoire compris entre 1 et 3 et donc de simuler le lancer du dé.*
   – *les variables `tirage_pour_A`, `tirage_pour_B` et `tirage_pour_C` représentent les numéros de place de parking attribués aux trois cadres ;*
   – *La variable `nbde1sortis_pour_C` permet de compter le nombre de fois où le cadre C se voit attribué la place de parking numéro 1.*

```
 1: VARIABLES
 2: i EST_DU_TYPE NOMBRE
 3: tirage_pour_A EST_DU_TYPE NOMBRE
 4: tirage_pour_B EST_DU_TYPE NOMBRE
 5: tirage_pour_C EST_DU_TYPE NOMBRE
 6: nbde1sortis_pour_C EST_DU_TYPE NOMBRE
 7: frequence EST_DU_TYPE NOMBRE
 8: DEBUT_ALGORITHME
 9:     nbde1sortis_pour_C PREND_LA_VALEUR 0
10:     POUR i ALLANT_DE 1 A 100000
11:         DEBUT_POUR
12:         tirage_pour_A PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,3)
13:         tirage_pour_B PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,3)
14:         TANT_QUE (tirage_pour_B..............) FAIRE
15:             DEBUT_TANT_QUE
16:             tirage_pour_B PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,3)
17:             FIN_TANT_QUE
18:         tirage_pour_C PREND_LA_VALEUR 6-tirage_pour_A-tirage_pour_B
19:         SI (tirage_pour_C==1) ALORS
20:             DEBUT_SI
21:             nbde1sortis_pour_C PREND_LA_VALEUR nbde1sortis_pour_C+1
22:             FIN_SI
23:         FIN_POUR
24:     frequence PREND_LA_VALEUR nbde1sortis_pour_C/100000
25:     AFFICHER frequence
26: FIN_ALGORITHME
```

   a) Comment faut-il compléter la ligne 14 pour que l'algorithme suive exactement la démarche proposée par le directeur des ressources humaines.
   b) Expliquer le calcul effectué ligne 18.
3. En exécutant l'algorithme complet plusieurs fois, émettre une conjecture sur la probabilité d'attribution de la place de parking numéro 1 au cadre C.

Partie B : *Étude théorique*
La démarche proposée par le directeur des ressources humaines peut se modéliser à l'aide l'arbre pondéré ci-dessous où les trois niveaux représentent la situation pour les cadres A, B et C (de gauche à droite).



1. Compléter l'arbre pondéré en y indiquant les probabilités sur les différentes branches.
   (remarque : le numéro du cadre A étant sorti, le cadre B a autant de chances de se voir attribuer un des deux numéros disponibles que l'autre)
2. En déduire la probabilité que le cadre C se voit attribuer la place de parking numéro 1.
   Le cadre C avait-il raison de se sentir désavantagé ?
3. Quelle est la probabilité que le cadre B se voit attribuer la place de parking numéro 1 ?
   La démarche proposée par le directeur des ressources humaines est-elle équitable ?

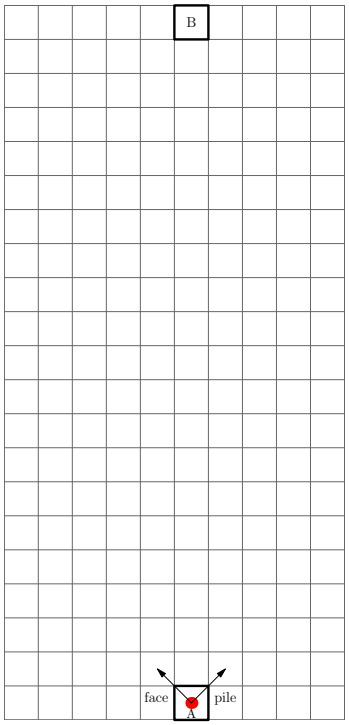
Fiche élève 3B

Un jeu consiste à essayer d'amener un jeton de la case A à la case B (distante verticalement de 20 cases) sur une grille (voir figure ci-dessous) en lançant 20 fois de suite une pièce selon le principe suivant :

– si la pièce donne « pile », on avance le jeton diagonalement d'une case vers le haut et vers la droite ;

– si la pièce donne « face », on avance le jeton diagonalement d'une case vers le haut et vers la gauche ;

On cherche à établir la probabilité d'atteindre la case B.



Partie A : *Simulation avec un algorithme*

1. Pour réussir à atteindre la case B, combien de fois faut-il tomber sur « pile » et « face » lors des 20 lancers consécutifs de la pièce ?

2. À l'aide de l'algorithme AlgoBox ci-dessous, on cherche à déterminer expérimentalement une estimation de la probabilité d'atteindre la case B en simulant 100 000 fois le principe du jeu.
   Remarques :
   – *la fonction* `ALGOBOX_ALEA_ENT(0,1)` *permet d'obtenir un entier pseudo-aléatoire égal à 0 ou à 1 et donc de simuler le lancer d'une pièce en associant ici 0 au fait d'obtenir « face » et 1 au fait d'obtenir « pile ».*
   – *La variable* `nb_piles` *permet de compter le nombre de fois où la pièce donne « pile » lors d'une partie.*
   – *La variable* `nb_jeux_gagnants` *permet de compter le nombre de fois où la case B est atteinte lors des* 100 000 *parties simulées.*

```
VARIABLES
i EST_DU_TYPE NOMBRE
nb_jeux_gagnants EST_DU_TYPE NOMBRE
lancer EST_DU_TYPE NOMBRE
nb_piles EST_DU_TYPE NOMBRE
frequence EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    nb_jeux_gagnants PREND_LA_VALEUR 0
    POUR i ALLANT_DE 1 A 100000
        DEBUT_POUR
        nb_piles PREND_LA_VALEUR 0
        POUR lancer ALLANT_DE 1 A ....
            DEBUT_POUR
            SI (ALGOBOX_ALEA_ENT(0,1)==1) ALORS
                DEBUT_SI
                nb_piles PREND_LA_VALEUR nb_piles+1
                FIN_SI
            FIN_POUR
        SI (.....................) ALORS
            DEBUT_SI
            nb_jeux_gagnants PREND_LA_VALEUR nb_jeux_gagnants+1
            FIN_SI
        FIN_POUR
    frequence PREND_LA_VALEUR nb_jeux_gagnants/100000
    AFFICHER frequence
FIN_ALGORITHME
```

3. Compléter les lignes 12 et 19 pour que l'algorithme corresponde à la simulation souhaitée.
4. En exécutant l'algorithme complet plusieurs fois, émettre une conjecture sur la probabilité d'atteindre la case B.

Partie B : *Étude théorique*
On note X le nombre de « pile » obtenus lors des 20 lancers consécutifs de la pièce.

1. Justifier que X suit une loi binomiale dont on donnera les paramètres.
2. En déduire la probabilité de réussir à atteindre la case B.

Fiche élève 3C

On considère l'expérience aléatoire consistant à choisir au hasard, et de façon indépendante, deux **réels** X et Y dans l'intervalle $[0\,;10]$.
On cherche à déterminer la probabilité que le maximum des deux réels soit inférieur à 5.

Partie A : *Simulation avec un algorithme*
À l'aide de l'algorithme AlgoBox ci-dessous, on cherche à déterminer expérimentalement une estimation de la probabilité cherchée en simulant 100 000 fois l'expérience aléatoire décrite ci-dessus.
Remarques :
– *l'instruction* `10*random()` *permet d'obtenir un réel pseudo-aléatoire compris entre 0 et 10.*
– *La variable* `nb_issues_favorables` *permet de compter le nombre de fois où le maximum est inférieur à 5 lors des* 100 000 *simulations.*

```
 1: VARIABLES
 2: X EST_DU_TYPE NOMBRE
 3: Y EST_DU_TYPE NOMBRE
 4: max EST_DU_TYPE NOMBRE
 5: i EST_DU_TYPE NOMBRE
 6: nb_issues_favorables EST_DU_TYPE NOMBRE
 7: frequence EST_DU_TYPE NOMBRE
 8: DEBUT_ALGORITHME
 9:    nb_issues_favorables PREND_LA_VALEUR 0
10:    POUR i ALLANT_DE 1 A 100000
11:       DEBUT_POUR
12:       X PREND_LA_VALEUR 10*random()
13:       Y PREND_LA_VALEUR 10*random()
14:       SI (X>Y) ALORS
15:          DEBUT_SI
16:          max PREND_LA_VALEUR ...........
17:          FIN_SI
18:       SINON
19:          DEBUT_SINON
20:          max PREND_LA_VALEUR ...........
21:          FIN_SINON
22:       SI (max.........) ALORS
23:          DEBUT_SI
24:          nb_issues_favorables PREND_LA_VALEUR nb_issues_favorables+1
25:          FIN_SI
26:       FIN_POUR
27:    frequence PREND_LA_VALEUR nb_issues_favorables/100000
28:    AFFICHER frequence
29: FIN_ALGORITHME
```

1. Compléter les lignes 16, 20 et 22 pour que l'algorithme corresponde à la simulation souhaitée.
2. En exécutant l'algorithme complet plusieurs fois, émettre une conjecture sur la probabilité que le maximum de X et Y soit inférieur à 5.
3. Proposer une condition dans l'algorithme qui permettrait de compter le nombre d'issues favorables sans avoir à déterminer le maximum de X et Y.

Partie B : *Étude théorique*
Le fait de choisir au hasard un réel dans l'intervalle $[0\,;10]$ peut se modéliser par une variable aléatoire suivant la loi uniforme sur $[0\,;10]$.
On peut donc supposer que X et Y suivent la loi uniforme sur $[0\,;10]$.

1. En utilisant les propriétés de la loi uniforme, donner les probabilités $p(X \leqslant 5)$ et $p(Y \leqslant 5)$.
2. Compléter la proposition suivante : « le maximum de X et Y est inférieur à 5 » équivaut à « X $\boxed{}$ » **et** « Y $\boxed{}$ ».
3. En utilisant l'indépendance des variables aléatoires X et Y, déterminer la probabilité que le maximum soit inférieur à 5.

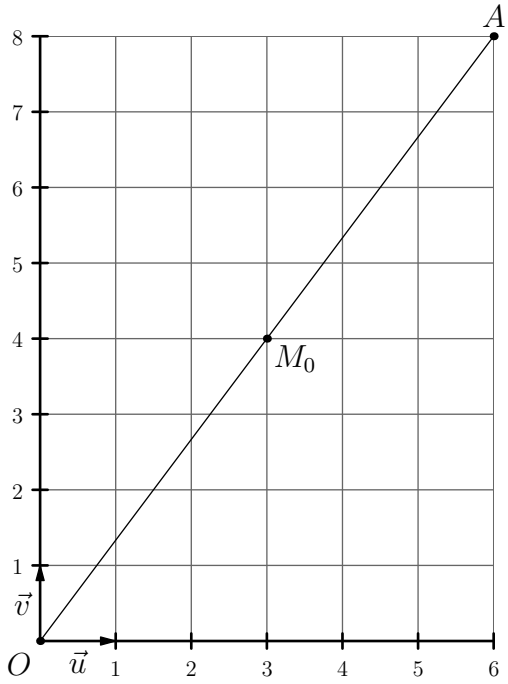
4 Complexes et géométrie

Fiche professeur 4A

– Fiche élève correspondante : page 25
– *Fichier AlgoBox associé (algorithme complet) :* `algo_4A.alg`
– *Contexte (TS) :* Détermination algorithmique d'un rang à partir duquel une suite définie comme une somme de distances dépasse un certain seuil.

Fiche élève 4A

Dans le plan complexe muni d'un repère orthonormé $\left(O,\vec{u},\vec{v}\right)$ d'unité 1 cm, on considère les points A d'affixe $z_A = 6+8\mathrm{i}$ et M_0 d'affixe $z_0 = \dfrac{z_A}{2}$.



Pour tout entier $n \geqslant 0$, on pose $z_{n+1} = \dfrac{z_A + z_n}{2}$ et on note M_n le point d'affixe z_n.

1. Montrer que, pour tout entier $n \geqslant 0$, on a $\overrightarrow{AM_{n+1}} = \dfrac{1}{2}\overrightarrow{AM_n}$.
2. Justifier par récurrence que, pour tout entier $n \geqslant 0$, le point M_n est sur la droite (OA) et que la distance AM_n est telle que $AM_n = 5\times\left(\dfrac{1}{2}\right)^n$.
3. En déduire que, pour tout entier $n \geqslant 1$, on a $AM_0 + AM_1 + \cdots + AM_{n-1} = 10\times\left(1-\left(\dfrac{1}{2}\right)^n\right)$.
4. Justifier que, pour tout entier $n \geqslant 0$, on a $|z_n| = 10 - AM_n$.
5. On considère, pour tout entier $n \geqslant 1$, la somme $S_n = |z_0| + |z_1| + \cdots + |z_{n-1}|$.
   a) Justifier que la suite (S_n) est croissante.
   b) À l'aide des questions 3 et 4, exprimer S_n en fonction de n.
   c) En déduire que $\lim\limits_{n\to+\infty} S_n = +\infty$.
   d) On cherche à déterminer le plus petit entier n tel que $S_n \geqslant 100$ à l'aide de l'algorithme AlgoBox ci-dessous. Compléter la ligne 7 pour que l'algorithme réponde à la question.

```
 1: VARIABLES
 2: S EST_DU_TYPE NOMBRE
 3: n EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:    S PREND_LA_VALEUR 5
 6:    n PREND_LA_VALEUR 1
 7:    TANT_QUE (S......) FAIRE
 8:       DEBUT_TANT_QUE
 9:       S PREND_LA_VALEUR S+10-5*pow(0.5,n)
10:       n PREND_LA_VALEUR n+1
11:       FIN_TANT_QUE
12:    AFFICHER n
13: FIN_ALGORITHME
```

Deuxième partie

Annexes

A

Structures algorithmiques de base avec AlgoBox

Les activités proposées dans cette annexe permettent de s'initier aux structures algorithmiques de base prévues au programme quelque soit la filière (aucun pré-requis mathématique spécifique n'est nécessaire).

A.1 Variables et affectations

Les variables en algorithmique

- Les variables algorithmiques peuvent servir à stocker des données de différents types, mais nous nous contenterons ici d'utiliser des variables du type `NOMBRE`.
- La valeur d'une variable peut changer au fil des instructions de l'algorithme.
- Les opérations sur les variables s'effectuent ligne après ligne et les unes après les autres.
- Quand l'ordinateur exécute une ligne du type `mavariable PREND_LA_VALEUR un calcul`, il effectue d'abord le calcul et stocke ensuite le résultat dans `mavariable`.

► Activité n°1
On considère l'algorithme suivant :

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: z EST_DU_TYPE NOMBRE
5: DEBUT_ALGORITHME
6:  | x PREND_LA_VALEUR 2
7:  | y PREND_LA_VALEUR 3
8:  | z PREND_LA_VALEUR x+y
9: FIN_ALGORITHME
```

Après exécution de l'algorithme :

- La variable `x` contient la valeur : ☐
- La variable `y` contient la valeur : ☐
- La variable `z` contient la valeur : ☐

► Activité n°2
On considère l'algorithme suivant :

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:  | x PREND_LA_VALEUR 2
5:  | x PREND_LA_VALEUR x+1
6: FIN_ALGORITHME
```

Après exécution de l'algorithme :

- La variable `x` contient la valeur : ☐

► Activité n°3
Ajoutons la ligne « `x PREND_LA_VALEUR 4*x` » à la fin du code précédent. Ce qui donne :

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:  | x PREND_LA_VALEUR 2
5:  | x PREND_LA_VALEUR x+1
6:  | x PREND_LA_VALEUR 4*x
7: FIN_ALGORITHME
```

Après exécution de l'algorithme :

- La variable `x` contient la valeur : ☐

► Activité n°4
On considère l'algorithme suivant :

```
1: VARIABLES
2: A EST_DU_TYPE NOMBRE
3: B EST_DU_TYPE NOMBRE
4: C EST_DU_TYPE NOMBRE
5: DEBUT_ALGORITHME
6:    A PREND_LA_VALEUR 5
7:    B PREND_LA_VALEUR 3
8:    C PREND_LA_VALEUR A+B
9:    B PREND_LA_VALEUR B+A
10:   A PREND_LA_VALEUR C
11: FIN_ALGORITHME
```

Après exécution de l'algorithme :

– La variable A contient la valeur : ☐

– La variable B contient la valeur : ☐

– La variable C contient la valeur : ☐

► Activité n°5
On considère l'algorithme suivant :

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: z EST_DU_TYPE NOMBRE
5: DEBUT_ALGORITHME
6:    LIRE x
7:    y PREND_LA_VALEUR x-2
8:    z PREND_LA_VALEUR -3*y-4
9:    AFFICHER z
10: FIN_ALGORITHME
```

On cherche maintenant à obtenir un algorithme équivalent sans utiliser la variable y. Compléter la ligne 6 dans l'algorithme ci-dessous pour qu'il réponde au problème.

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: z EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:    LIRE x
6:    z PREND_LA_VALEUR .....................
7:    AFFICHER z
8: FIN_ALGORITHME
```

A.2 Instructions conditionnelles

SI...ALORS...SINON

Comme nous l'avons vu ci-dessus, un algorithme permet d'exécuter une liste d'instructions les unes à la suite des autres. Mais on peut aussi "dire" à un algorithme de n'exécuter des instructions que si une certaine condition est remplie. Cela se fait grâce à la commande SI...ALORS :

```
SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
```

Il est aussi possible d'indiquer en plus à l'algorithme de traiter le cas où la condition n'est pas vérifiée. On obtient alors la structure suivante :

```
SI...ALORS
  DEBUT_SI
  ...
  FIN_SI
  SINON
    DEBUT_SINON
    ...
    FIN_SINON
```

► Activité n°6

On cherche à créer un algorithme qui demande un nombre à l'utilisateur et qui affiche la racine carrée de ce nombre s'il est positif. Compléter la ligne 6 dans l'algorithme ci-dessous pour qu'il réponde au problème.

```
 1: VARIABLES
 2: x EST_DU_TYPE NOMBRE
 3: racine EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   LIRE x
 6:   SI (.........) ALORS
 7:     DEBUT_SI
 8:     racine PREND_LA_VALEUR sqrt(x)
 9:     AFFICHER racine
10:     FIN_SI
11: FIN_ALGORITHME
```

► Activité n°7

On cherche à créer un algorithme qui demande à l'utilisateur d'entrer deux nombres (stockés dans les variables x et y) et qui affiche le plus grand des deux. Compléter les ligne 9 et 13 dans l'algorithme ci-dessous pour qu'il réponde au problème.

```
 1: VARIABLES
 2: x EST_DU_TYPE NOMBRE
 3: y EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   LIRE x
 6:   LIRE y
 7:   SI (x>y) ALORS
 8:     DEBUT_SI
 9:     AFFICHER .......
10:     FIN_SI
11:   SINON
12:     DEBUT_SINON
13:     AFFICHER ......
14:     FIN_SINON
15: FIN_ALGORITHME
```

► Activité n°8
On considère l'algorithme suivant :

```
1: VARIABLES
2: A EST_DU_TYPE NOMBRE
3: B EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   A PREND_LA_VALEUR 1
6:   B PREND_LA_VALEUR 3
7:   SI (A>0) ALORS
8:     DEBUT_SI
9:     A PREND_LA_VALEUR A+1
10:    FIN_SI
11:  SI (B>4) ALORS
12:    DEBUT_SI
13:    B PREND_LA_VALEUR B-1
14:    FIN_SI
15: FIN_ALGORITHME
```

Après exécution de l'algorithme :

– La variable A contient la valeur : ☐

– La variable B contient la valeur : ☐

► Activité n°9
On cherche à concevoir un algorithme correspondant au problème suivant :
– on demande à l'utilisateur d'entrer un nombre (qui sera représenté par la variable x)
– si le nombre entré est différent de 1, l'algorithme doit stocker dans une variable y la valeur de 1/(x-1) et afficher la valeur de y (note : la condition x différent de 1 s'exprime avec le code x!=1). On ne demande pas de traiter le cas contraire.

Compléter l'algorithme ci-dessous pour qu'il réponde au problème.

```
1: VARIABLES
2: x EST_DU_TYPE NOMBRE
3: y EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   LIRE ......
6:   SI (........) ALORS
7:     DEBUT_SI
8:     ....... PREND_LA_VALEUR .......
9:     AFFICHER .......
10:    FIN_SI
11: FIN_ALGORITHME
```

A.3 Boucles

Boucles POUR...DE...A

- Les boucles permettent de répéter des instructions autant de fois que l'on souhaite.
- Lorsqu'on connait par avance le nombre de fois que l'on veut répéter les instructions, on utilise une boucle du type POUR...DE...A dont la structure est la suivante :

```
POUR...ALLANT_DE...A...
  DEBUT_POUR
  ...
  FIN_POUR
```

- Exemple : l'algorithme ci-dessous permet d'afficher la racine carrée de tous les entiers de 1 jusqu'à 50.

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: racine EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   POUR n ALLANT_DE 1 A 50
 6:     DEBUT_POUR
 7:     racine PREND_LA_VALEUR sqrt(n)
 8:     AFFICHER racine
 9:     FIN_POUR
10: FIN_ALGORITHME
```

  La variable n est appelée « compteur de la boucle ».
- Remarques :
  - La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
  - Dans AlgoBox, cette variable est automatiquement augmentée de 1 à chaque fois.
  - On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais les instructions comprises entre DEBUT_POUR et FIN_POUR ne doivent en aucun cas modifier la valeur de la variable qui sert de compteur.

► Activité n°10

On cherche à concevoir un algorithme qui affiche, grâce à une boucle POUR...DE...A, les résultats des calculs suivants : 8*1 ; 8*2 ; 8*3 ; 8*4 ; ... jusqu'à 8*10 .
La variable n sert de compteur à la boucle et la variable produit sert à stocker et afficher les résultats. Compléter les lignes 5 et 7 dans l'algorithme ci-dessous pour qu'il réponde au problème :

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: produit EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   POUR n ALLANT_DE .... A .....
 6:     DEBUT_POUR
 7:     produit PREND_LA_VALEUR .......
 8:     AFFICHER produit
 9:     FIN_POUR
10: FIN_ALGORITHME
```

► Activité n°11

On considère l'algorithme suivant :

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: somme EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:   somme PREND_LA_VALEUR 0
 6:   POUR n ALLANT_DE 1 A 100
 7:     DEBUT_POUR
 8:     somme PREND_LA_VALEUR somme+n
 9:     FIN_POUR
10:   AFFICHER somme
11: FIN_ALGORITHME
```

Compléter les phrases suivantes :

- Après exécution de la ligne 5, La variable `somme` contient la valeur :
- Lorsque le compteur `n` de la boucle vaut 1 et après exécution du calcul ligne 8, la variable `somme` vaut :
- Lorsque le compteur `n` de la boucle vaut 2 et après exécution du calcul ligne 8, la variable `somme` vaut :
- Lorsque le compteur `n` de la boucle vaut 3 et après exécution du calcul ligne 8, la variable `somme` vaut :

Que permet de calculer cet algorithme ?

► Activité n°12

Compléter les lignes 6 et 8 de l'algorithme ci-dessous pour qu'il permette de calculer la somme $5^2 + 6^2 + 7^2 + \cdots + 24^2 + 25^2$.

```
 1: VARIABLES
 2: n EST_DU_TYPE NOMBRE
 3: somme EST_DU_TYPE NOMBRE
 4: DEBUT_ALGORITHME
 5:     somme PREND_LA_VALEUR 0
 6:     POUR n ALLANT_DE .... A ......
 7:         DEBUT_POUR
 8:         somme PREND_LA_VALEUR somme+......
 9:         FIN_POUR
10:     AFFICHER somme
11: FIN_ALGORITHME
```

Boucles TANT QUE...

– Il n'est pas toujours possible de connaitre par avance le nombre de répétitions nécessaires à un calcul. Dans ce cas là, il est possible d'avoir recours à la structure `TANT QUE...` qui se présente de la façon suivante :

```
TANT_QUE...FAIRE
  DEBUT_TANT_QUE
  ...
  FIN_TANT_QUE
```

Cette structure de boucle permet de répéter une série d'instructions (comprises entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE`) tant qu'*une certaine condition est vérifiée*.

– Exemple : Comment savoir ce qu'il reste si on enlève 25 autant de fois que l'on peut au nombre 583 ? Pour cela on utilise une variable `n`, qui contient 583 au début, à laquelle on enlève 25 tant que c'est possible, c'est à dire tant que n est supérieur ou égal à 25.

```
1: VARIABLES
2: n EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   n PREND_LA_VALEUR 583
5:   TANT_QUE (n>=25) FAIRE
6:     DEBUT_TANT_QUE
7:     n PREND_LA_VALEUR n-25
8:     FIN_TANT_QUE
9:   AFFICHER n
10: FIN_ALGORITHME
```

– Remarques :
  – Si la condition du TANT QUE... est fausse dès le début, les instructions entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` ne sont jamais exécutées (la structure TANT QUE ne sert alors strictement à rien).
  – Il est indispensable de s'assurer que la condition du TANT QUE... finisse par être vérifiée (le code entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme ne pourra pas fonctionner.

► Activité n°13

Un individu a emprunté à un ami une somme de 2500 euros (prêt sans intérêts). Pour rembourser son ami, il prévoit de lui remettre 110 euros par mois. Mais comme cela ne correspond pas à un nombre pile de mois, il se demande quel sera le montant à rembourser le dernier mois.
Compléter la ligne 5 dans l'algorithme ci-dessous pour qu'il réponde au problème :

```
1: VARIABLES
2: montant EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   montant PREND_LA_VALEUR 2500
5:   TANT_QUE (.............) FAIRE
6:     DEBUT_TANT_QUE
7:     montant PREND_LA_VALEUR montant-110
8:     FIN_TANT_QUE
9:   AFFICHER montant
10: FIN_ALGORITHME
```

► Activité n°14

On cherche à connaître le plus petit entier N tel que 2^N soit supérieur ou égal à 10000. Pour résoudre ce problème de façon algorithmique :
– On utilise une variable `N` à laquelle on donne au début la valeur 1.
– On augmente de 1 la valeur de `N` tant que 2^N n'est pas supérieur ou égal à 10000.
Une structure TANT QUE est particulièrement adaptée à ce genre de problème car on ne sait pas a priori combien de calculs seront nécessaires.
Compléter les lignes 5 et 7 de l'algorithme ci-dessous pour qu'il réponde au problème :
(remarque : `pow(2,N)` est le code AlgoBox pour calculer 2^N)

```
1: VARIABLES
2: N EST_DU_TYPE NOMBRE
3: DEBUT_ALGORITHME
4:   N PREND_LA_VALEUR 1
5:   TANT_QUE (pow(2,N).......) FAIRE
6:     DEBUT_TANT_QUE
7:     N PREND_LA_VALEUR .......
8:     FIN_TANT_QUE
9:   AFFICHER N
10: FIN_ALGORITHME
```

► Activité n°15

On considère le problème suivant :

- On lance une balle d'une hauteur initiale de 300 cm.
- On suppose qu'à chaque rebond, la balle perd 10% de sa hauteur (la hauteur est donc multipliée par 0.9 à chaque rebond).
- On cherche à savoir le nombre de rebonds nécessaire pour que la hauteur de la balle soit inférieure ou égale à 10 cm.

Compléter les lignes 7 et 10 de l'algorithme ci-dessous pour qu'il réponde au problème.

```
1: VARIABLES
2: nombre_rebonds EST_DU_TYPE NOMBRE
3: hauteur EST_DU_TYPE NOMBRE
4: DEBUT_ALGORITHME
5:   nombre_rebonds PREND_LA_VALEUR 0
6:   hauteur PREND_LA_VALEUR 300
7:   TANT_QUE (hauteur......) FAIRE
8:     DEBUT_TANT_QUE
9:     nombre_rebonds PREND_LA_VALEUR nombre_rebonds+1
10:    hauteur PREND_LA_VALEUR ...........
11:    FIN_TANT_QUE
12:  AFFICHER nombre_rebonds
13: FIN_ALGORITHME
```

Mémento sur l'utilisation d'AlgoBox

B.1 Équivalence entre « pseudo-codes »

Résumé des différences entre le pseudo-code utilisé par AlgoBox et celui que l'on peut rencontrer dans les manuels.

B.1.1 Entrée des données

Pseudo-code AlgoBox	Variantes
`LIRE...`	`Saisir...` `Entrer...`

B.1.2 Affichage des données

Pseudo-code AlgoBox	Variantes
`AFFICHER...`	`Écrire...`

B.1.3 Affecter une valeur à une variable

Pseudo-code AlgoBox	Variantes

`A PREND_LA_VALEUR 3`	`Affecter 3 à A`
	`A:=3`
	`A←3`

B.1.4 Structure SI ALORS

Pseudo-code AlgoBox	Variantes

`SI...ALORS`
`  DEBUT_SI`
`  ...`
`  FIN_SI`
	`Si...alors`
	`  ...`
	`FinSi`
	
	`Si...`
	`Alors...`
	`FinSi`

Pseudo-code AlgoBox	Variantes

`SI...ALORS`
`  DEBUT_SI`
`  ...`
`  FIN_SI`
`  SINON`
`  DEBUT_SINON`
`  ...`
`  FIN_SINON`
	`Si...alors`
	`  ...`
	`  sinon`
	`  ...`
	`FinSi`
	
	`Si...`
	`Alors...`
	`Sinon...`
	`FinSi`

B.1.5 Boucle POUR...

Pseudo-code AlgoBox	Variantes

`POUR...ALLANT_DE...A...`
`  DEBUT_POUR`
`  ...`
`  FIN_POUR`
	`Pour...de...jusqu'à...`
	`  ...`
	`FinPour`
	
	`Pour...variant de...à...`
	`  ...`
	`FinPour`

B.1.6 Structure TANT QUE...

Pseudo-code AlgoBox	Variantes

`TANT_QUE...FAIRE`
`  DEBUT_TANT_QUE`
`  ...`
`  FIN_TANT_QUE`
	`Tant que...`
	`  ...`
	`FinTantQue`

B.2 Les problèmes de syntaxe

B.2.1 Les erreurs de syntaxe les plus courantes

– *Erreur classique n°1 :* utiliser la virgule au lieu du point comme séparateur décimal
– *Erreur classique n°2 :* utiliser la syntaxe "`SI x=y`" au lieu de "`SI x==y`" pour vérifier une égalité dans une condition
– *Erreur classique n°3 :* utiliser la syntaxe `x^y` au lieu de `pow(x,y)` pour les puissances.

B.2.2 Syntaxe des opérations mathématiques

Opération mathématique	syntaxe AlgoBox
$\sqrt{x}$	`sqrt(x)`
x^y	`pow(x,y)`
$\cos(x)$ (x en radians)	`cos(x)`
$\sin(x)$ (x en radians)	`sin(x)`
$\tan(x)$ (x en radians)	`tan(x)`
e^x	`exp(x)`
$\ln x$	`log(x)`
$\lvert x \rvert$	`abs(x)`
Partie entière de x	`floor(x)`
Nombre pseudo-aléatoire compris entre 0 et 1	`random()`
Reste de la division euclidienne de n par p	`n%p`
π	`Math.PI`
$\arccos(x)$	`acos(x)`
$\arcsin(x)$	`asin(x)`
$\arctan(x)$	`atan(x)`
Valeur approchée de x à 10^{-n} près	`ALGOBOX_ARRONDIR(x,n)`

B.2.3 Syntaxe pour les conditions

Condition logique	syntaxe AlgoBox
$x = y$	`x==y`
$x \neq y$	`x!=y`
$x < y$	`x<y`
$x \leqslant y$	`x<=y`
$x > y$	`x>y`
$x \geqslant y$	`x>=y`
condition1 **ou** condition2	`condition1 OU condition2`
condition1 **et** condition2	`condition1 ET condition2`

Remarque : il est possible d'inclure des opérations mathématiques dans les conditions. (exemple : `pow(2,n)<100`)

B.2.4 Syntaxe pour les fonctions statistiques et les opérations sur les listes

Somme des termes d'une liste	`ALGOBOX_SOMME(nom_liste,rang_premier_terme,rang_dernier_terme)`
Moyenne	`ALGOBOX_MOYENNE(nom_liste,rang_premier_terme,rang_dernier_terme)`
Variance	`ALGOBOX_VARIANCE(nom_liste,rang_premier_terme,rang_dernier_terme)`
Écart-type	`ALGOBOX_ECART_TYPE(nom_liste,rang_premier_terme,rang_dernier_terme)`
Médiane	`ALGOBOX_MEDIANE(nom_liste,rang_premier_terme,rang_dernier_terme)`
Q1 (version calculatrice)	`ALGOBOX_QUARTILE1(nom_liste,rang_premier_terme,rang_dernier_terme)`
Q3 (version calculatrice)	`ALGOBOX_QUARTILE3(nom_liste,rang_premier_terme,rang_dernier_terme)`
Q1 (version « officielle »)	`ALGOBOX_QUARTILE1_BIS(nom_liste,rang_premier_terme,rang_dernier_terme)`
Q3 (version « officielle »)	`ALGOBOX_QUARTILE3_BIS(nom_liste,rang_premier_terme,rang_dernier_terme)`
Minimum d'une liste	`ALGOBOX_MINIMUM(nom_liste,rang_premier_terme,rang_dernier_terme)`
Maximum d'une liste	`ALGOBOX_MAXIMUM(nom_liste,rang_premier_terme,rang_dernier_terme)`
Rang du minimum	`ALGOBOX_POS_MINIMUM(nom_liste,rang_premier_terme,rang_dernier_terme)`
Rang du maximum	`ALGOBOX_POS_MAXIMUM(nom_liste,rang_premier_terme,rang_dernier_terme)`

B.2.5 Fonctions concernant les probabilités

entier pseudo-aléatoire entre p et n	`ALGOBOX_ALEA_ENT(p,n)`
$\binom{n}{p}$ $(n \leqslant 100)$	`ALGOBOX_COEFF_BINOMIAL(n,p)`
$p(X = k)$ pour la loi binomiale $(n \leqslant 100)$	`ALGOBOX_LOI_BINOMIALE(n,p,k)`
$p(X < x)$ pour la loi normale centrée réduite	`ALGOBOX_LOI_NORMALE_CR(x)`
$p(X < x)$ pour la loi normale	`ALGOBOX_LOI_NORMALE(esp,ecart,x)`
x tel $p(X < x) = p$ pour la loi normale centrée réduite	`ALGOBOX_INVERSE_LOI_NORMALE_CR(p)`
x tel $p(X < x) = p$ pour la loi normale	`ALGOBOX_INVERSE_LOI_NORMALE(esp,ecart,p)`
$n!$ $(n \leqslant 69)$	`ALGOBOX_FACTORIELLE(n)`

B.2.6 Fonctions concernant les chaines de caractères

Remarque : le contenu d'une chaîne doit être encadré par des guillemets.
(exemple : `machaine prend la valeur "bonjour"`)

concaténation de deux chaines A et B	`A+B`
sous-chaine de n caractères à partir de la position p	`machaine.substr(p,n)`
transformation d'un nombre n en chaine	`n.toString()`
transformation d'une chaine en entier	`parseInt(machaine)`
transformation d'une chaine en réel	`parseFloat(machaine)`
longueur d'une chaine	`machaine.length`
code ASCII du caractère situé à la position p	`machaine.charCodeAt(p)`
chaine de code ASCII p	`String.fromCharCode(p)`

B.3 Fonctionnement d'AlgoBox

B.3.1 Les deux règles fondamentales

1. Toute variable doit d'abord être déclarée avant de pouvoir être utilisée. La première chose à faire avant de concevoir un algorithme est d'ailleurs de lister toutes les variables qui seront nécessaires.
2. Une nouvelle instruction ne peut s'insérer que sur une ligne vierge.

B.3.2 Les variables

Attention

Le nom des variables ne doit pas contenir d'espaces (que l'on peut remplacer par le caractère _), ni d'accents, ni de caractères spéciaux. On ne peut pas non plus utiliser certains termes réservés : par exemple, une variable ne peut pas être appelée `NOMBRE`. Il est par contre conseillé d'utiliser des noms de variables explicites (mêmes longs) pour rendre les algorithmes plus clairs.

B.3.3 Les listes de nombres

Il est possible d'entrer directement les termes d'une liste. Pour cela, il suffit d'utiliser l'instruction `LIRE maliste[1]` (1 représentant le premier rang des termes de la liste que l'on veut entrer). Lors de l'exécution de l'algorithme, il suffira alors d'entrer toutes les valeurs souhaitées (dans l'ordre) en les séparant par : .

B.3.4 Boucle POUR...DE...A

- On n'utilise ce genre de boucles que si on connaît à l'avance le nombre de répétitions à effectuer.
- La variable servant de compteur pour la boucle doit être du type NOMBRE et doit être déclarée préalablement (comme toutes les variables).
- Dans AlgoBox, cette variable est automatiquement augmentée de 1 à chaque fois.

Attention

- On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais les instructions comprises entre `DEBUT_POUR` et `FIN_POUR` ne doivent en aucun cas modifier la valeur de la variable qui sert de compteur.
- Le nombre d'itérations sur AlgoBox est limité à 500 000. En cas de dépassement, l'algorithme s'arrête et le message « ***Algorithme interrompu suite à une erreur*** » est affiché.
- Si les instructions à répéter comportent l'affichage d'une variable ou un tracé graphique, il faut limiter le nombre d'itérations à moins d'un millier (ou guère plus) : sans quoi, l'exécution du programme risque de prendre beaucoup trop de temps. Par contre, s'il n'y que des calculs à répéter on peut pousser le nombre d'itérations plus loin.

B.3.5 Structure TANT QUE

- Cette structure sert à répéter des instructions que l'on connaisse ou non par avance le nombre d'itérations à effectuer.
- Si la condition du TANT QUE... est fausse dès le début, les instructions entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` ne sont jamais exécutées (la structure TANT QUE ne sert alors strictement à rien).

Attention

- Il est indispensable de s'assurer que la condition du TANT QUE... finisse par être vérifiée (le code entre `DEBUT_TANT_QUE` et `FIN_TANT_QUE` doit rendre vraie la condition tôt ou tard), sans quoi l'algorithme va se lancer dans une « boucle infinie ».
- Afin d'éviter les boucles infinies, le nombre d'itérations est là aussi limité à 500 000. En cas de dépassement, l'algorithme s'arrête et le message « ***Algorithme interrompu suite à une erreur*** » est affiché.
- Si les instructions à répéter comportent l'affichage d'une variable ou un tracé graphique, il faut limiter le nombre d'itérations à moins d'un millier (ou guère plus) : sans quoi, l'exécution du programme risque de prendre beaucoup trop de temps. Par contre, s'il n'y que des calculs à répéter on peut pousser le nombre d'itérations plus loin.

B.3.6 Utilisation de l'onglet « Utiliser une fonction numérique »

En activant l'option *"Utiliser une fonction"* dans l'onglet *"Utiliser une fonction numérique"*, on peut utiliser l'image de n'importe quel nombre (ou variable de type nombre) par la fonction notée `F1` dans le code de l'algorithme. Il suffit pour cela d'entrer l'expression de `F1(x)` en fonction de `x` dans le champ prévu pour cela. Pour utiliser l'image d'un nombre `nb` par la fonction `F1` dans l'algorithme, il suffit d'utiliser le code : `F1(nb)` (cela peut se faire dans une affectation ou dans une expression conditionnelle). Cette option est particulièrement utile quand un algorithme nécessite le calcul de plusieurs images par une même fonction.
Un exemple classique est celui de la dichotomie :

```
VARIABLES
precision EST_DU_TYPE NOMBRE
a EST_DU_TYPE NOMBRE
b EST_DU_TYPE NOMBRE
m EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    a PREND_LA_VALEUR ...
    b PREND_LA_VALEUR ...
    LIRE precision
    TANT_QUE (b-a>precision) FAIRE
        DEBUT_TANT_QUE
        m PREND_LA_VALEUR (a+b)/2
        SI (F1(m)*F1(b)>0) ALORS
            DEBUT_SI
            b PREND_LA_VALEUR m
            FIN_SI
        SINON
            DEBUT_SINON
            a PREND_LA_VALEUR m
            FIN_SINON
        AFFICHER a
        AFFICHER " < solution < "
        AFFICHER b
        FIN_TANT_QUE
FIN_ALGORITHME
```

B.3.7 Utilisation de l'onglet « Dessiner dans un repère »

En activant l'option *"Utiliser un repère"* dans l'onglet *"Dessiner dans un repère"*, un repère graphique est automatiquement ajouté dans la fenêtre de test de l'algorithme. Il est alors possible d'inclure dans le code de l'algorithme des instructions pour tracer des points et des segments dans ce repère en utilisant les boutons *"Ajouter TRACER POINT"* et *"Ajouter TRACER SEGMENT"*.

Attention

La « fenêtre » du repère est définie lors de la première utilisation des instructions `TRACER_POINT` et `TRACER_SEGMENT`. Si la fenêtre doit dépendre de la valeur de certaines variables, il faut s'assurer que celles-ci sont bien définies avant le premier tracé.

Exemple : représentation graphique d'une fluctuation d'échantillonnage (100 simulations de 1000 lancers d'une pièce)

```
VARIABLES
simulation EST_DU_TYPE NOMBRE
lancer EST_DU_TYPE NOMBRE
nb_de_piles EST_DU_TYPE NOMBRE
frequence EST_DU_TYPE LISTE
moy EST_DU_TYPE NOMBRE
ecart_type EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
| POUR simulation ALLANT_DE 1 A 100
| | DEBUT_POUR
| | nb_de_piles PREND_LA_VALEUR 0
| | POUR lancer ALLANT_DE 1 A 1000
| | | DEBUT_POUR
| | | SI (random()<0.5) ALORS
| | | | DEBUT_SI
| | | | nb_de_piles PREND_LA_VALEUR nb_de_piles+1
| | | | FIN_SI
| | | FIN_POUR
| | frequence[simulation] PREND_LA_VALEUR nb_de_piles/1000
| | TRACER_POINT (simulation,frequence[simulation])
| | FIN_POUR
| moy PREND_LA_VALEUR ALGOBOX_MOYENNE(frequence,1,100)
| TRACER_SEGMENT (1,moy)->(100,moy)
| ecart_type PREND_LA_VALEUR ALGOBOX_ECART_TYPE(frequence,1,100)
| TRACER_SEGMENT (1,moy+2*ecart_type)->(100,moy+2*ecart_type)
| TRACER_SEGMENT (1,moy-2*ecart_type)->(100,moy-2*ecart_type)
FIN_ALGORITHME
```

```
Xmin: 1 ; Xmax: 100 ; Ymin: 0.4 ; Ymax: 0.6 ; GradX: 10 ; GradY: 0.01
```

B.3.8 Utilisation de l'onglet « Fonction avancée »

En activant l'option *"Utiliser la fonction F2..."* dans l'onglet *"Fonction avancée"*, on peut définir :
- une fonction définie par « morceaux » ;
- une fonction dépendant de plusieurs paramètres (contrairement à la fonction `F1` qui ne doit dépendre que de x) ;
- une fonction définie de façon récursive.

Pour cela, il faut :
- préciser les paramètres dont dépend la fonction ;
- ajouter une à une les conditions permettant de définir la fonction ;
- indiquer ce que doit retourner la fonction quand aucune des conditions n'est remplie.

Exemple avec le calcul du pgcd de deux entiers (méthode récursive) :

```
VARIABLES
p EST_DU_TYPE NOMBRE
q EST_DU_TYPE NOMBRE
pgcd EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
| LIRE p
| LIRE q
| pgcd PREND_LA_VALEUR F2(p,q)
| AFFICHER pgcd
FIN_ALGORITHME
```

```
fonction F2(p,q):
SI (p==0) RENVOYER q
SI (q==0) RENVOYER p
SI (q<=p) RENVOYER F2(p-q,q)
Dans les autres cas, RENVOYER F2(p,q-p)
```

B.3.9 Récupération facile d'un code AlgoBox dans un traitement de texte

Pour copier un code AlgoBox facilement sans passer par les commandes d'exportation du menu *Fichier*, il suffit de :

– lancer la fenêtre de test de l'algorithme ;
– sélectionner le code à copier dans la page web de test (la partie supérieure de la fenêtre) ;
– faire « glisser » le code sélectionné vers son traitement de texte.

B.4 Quelques techniques classiques

B.4.1 Diviseur ?

– Condition pour vérifier si un entier P est un diviseur d'un entier N : `N%P==0`

B.4.2 Entier pair ou impair ?

– Condition pour vérifier si un entier N est pair : `N%2==0`
– Condition pour vérifier si un entier N est impair : `N%2!=0` (autre condition possible : `N%2==1`)

B.4.3 Entier pseudo-aléatoire compris entre 1 et N

– Pour obtenir un entier pseudo-aléatoire compris entre 1 et N :
`... PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,N)` ou `... PREND_LA_VALEUR floor(N*random()+1)`
– Exemple pour la face d'un dé : `... PREND_LA_VALEUR ALGOBOX_ALEA_ENT(1,6)`

B.4.4 « Balayage » d'un intervalle

Deux méthodes peuvent être utilisées pour « balayer » un intervalle $[a\,;b]$ (a et b faisant partie du traitement).
Première méthode en entrant le nombre de subdivisions et en utilisant une boucle POUR... :

```
1: VARIABLES
2: i EST_DU_TYPE NOMBRE
3: nbsubdivisions EST_DU_TYPE NOMBRE
4: pas EST_DU_TYPE NOMBRE
5: x EST_DU_TYPE NOMBRE
6: a EST_DU_TYPE NOMBRE
7: b EST_DU_TYPE NOMBRE
8: DEBUT_ALGORITHME
9:    LIRE a
10:   LIRE b
11:   LIRE nbsubdivisions
12:   pas PREND_LA_VALEUR (b-a)/nbsubdivisions
13:   POUR i ALLANT_DE 0 A nbsubdivisions
14:      DEBUT_POUR
15:      x PREND_LA_VALEUR a+i*pas
16:      traitement....
17:      FIN_POUR
18: FIN_ALGORITHME
```

Deuxième méthode en entrant directement le pas et en utilisant une structure TANT QUE... :

```
VARIABLES
pas EST_DU_TYPE NOMBRE
x EST_DU_TYPE NOMBRE
a EST_DU_TYPE NOMBRE
b EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE a
    LIRE b
    LIRE pas
    x PREND_LA_VALEUR a
    TANT_QUE (x<=b) FAIRE
        DEBUT_TANT_QUE
        traitement....
        x PREND_LA_VALEUR x+pas
        FIN_TANT_QUE
FIN_ALGORITHME
```

B.4.5 Suites numériques

D'un strict point de vue programmation, l'utilisation d'une liste pour manipuler les termes d'une suite dans un algorithme n'est guère optimal. Par contre, d'un point de vue pédagogique, la correspondance étroite entre terme d'une suite et terme d'une liste AlgoBox peut permettre de conforter la compréhension par les élèves du formalisme sur les suites numériques.
Pour modéliser une suite (U_n) à l'aide d'un algorithme AlgoBox, on peut utiliser une variable de type `LISTE` notée `U`.
Ainsi :

- le terme U_0 de la suite sera représenté par `U[0]` dans l'algorithme ;
- le terme U_1 de la suite sera représenté par `U[1]` dans l'algorithme ;
- etc...
- le terme U_n de la suite sera représenté par `U[n]` dans l'algorithme ;
- le terme U_{n+1} de la suite sera représenté par `U[n+1]` dans l'algorithme ;

Exemple avec une suite « récurrente » :
Sans utiliser de liste :

```
VARIABLES
n EST_DU_TYPE NOMBRE
U EST_DU_TYPE NOMBRE
i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE n
    U PREND_LA_VALEUR 1
    POUR i ALLANT_DE 0 A n-1
        DEBUT_POUR
        U PREND_LA_VALEUR 1+2*U
        FIN_POUR
    AFFICHER U
FIN_ALGORITHME
```

En utilisant une liste AlgoBox :

```
VARIABLES
n EST_DU_TYPE NOMBRE
U EST_DU_TYPE LISTE
i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE n
    U[0] PREND_LA_VALEUR 1
    POUR i ALLANT_DE 0 A n-1
        DEBUT_POUR
        U[i+1] PREND_LA_VALEUR 1+2*U[i]
        FIN_POUR
    AFFICHER U[n]
FIN_ALGORITHME
```

B.4.6 Échanger le contenu de deux variables

Pour échanger le contenu de deux variables `A` et `B`, il suffit d'utiliser une troisième variable `temp` :

```
temp PREND_LA_VALEUR A
A PREND_LA_VALEUR B
B PREND_LA_VALEUR temp
```

Exemple : tri d'une liste de valeurs par échange du minimum

```
 1: VARIABLES
 2: listenombre EST_DU_TYPE LISTE
 3: temp EST_DU_TYPE NOMBRE
 4: i EST_DU_TYPE NOMBRE
 5: min EST_DU_TYPE NOMBRE
 6: nbtermes EST_DU_TYPE NOMBRE
 7: DEBUT_ALGORITHME
 8:   LIRE nbtermes
 9:   //Entrer les "nbtermes" valeurs sous la forme valeur1:valeur2:etc...
10:   LIRE listenombre[1]
11:   POUR i ALLANT_DE 1 A nbtermes-1
12:     DEBUT_POUR
13:     min PREND_LA_VALEUR ALGOBOX_POS_MINIMUM(listenombre,i+1,nbtermes)
14:     SI (listenombre[i]>listenombre[min]) ALORS
15:       DEBUT_SI
16:       temp PREND_LA_VALEUR listenombre[min]
17:       listenombre[min] PREND_LA_VALEUR listenombre[i]
18:       listenombre[i] PREND_LA_VALEUR temp
19:       FIN_SI
20:     FIN_POUR
21:   POUR i ALLANT_DE 1 A nbtermes
22:     DEBUT_POUR
23:     AFFICHER listenombre[i]
24:     AFFICHER " "
25:     FIN_POUR
26: FIN_ALGORITHME
```

B.4.7 Afficher un message contenant du texte et des nombres

Il suffit pour cela d'utiliser une variable du type `CHAINE`.
Exemple :

```
 1: VARIABLES
 2: a EST_DU_TYPE NOMBRE
 3: b EST_DU_TYPE NOMBRE
 4: message EST_DU_TYPE CHAINE
 5: somme EST_DU_TYPE NOMBRE
 6: DEBUT_ALGORITHME
 7:   LIRE a
 8:   LIRE b
 9:   somme PREND_LA_VALEUR a+b
10:   message PREND_LA_VALEUR a.toString()+" + "+b.toString()+" est égal à "+somme.toString()
11:   AFFICHER message
12: FIN_ALGORITHME
```

Résultat :

```
***Algorithme lancé***
Entrer a : 4
Entrer b : 7
4 + 7 est égal à 11
***Algorithme terminé***
```

C Algorithmes supplémentaires

C.1 Répétition d'épreuves et loi normale

Cet algorithme illustre le lien entre la loi normale de paramètres $\mu = 18$ et $\sigma = 3$ et le phénomène aléatoire consistant à effectuer 10000 sommes de 36 entiers aléatoires égaux à 0 ou à 1.

```
VARIABLES
i EST_DU_TYPE NOMBRE
j EST_DU_TYPE NOMBRE
somme EST_DU_TYPE NOMBRE
cumul EST_DU_TYPE LISTE
DEBUT_ALGORITHME
    POUR j ALLANT_DE 0 A 36
        DEBUT_POUR
        cumul[j] PREND_LA_VALEUR 0
        FIN_POUR
    POUR i ALLANT_DE 1 A 10000
        DEBUT_POUR
        somme PREND_LA_VALEUR 0
        POUR j ALLANT_DE 1 A 36
            DEBUT_POUR
            somme PREND_LA_VALEUR somme+ALGOBOX_ALEA_ENT(0,1)
            FIN_POUR
        cumul[somme] PREND_LA_VALEUR cumul[somme]+1
        FIN_POUR
    POUR j ALLANT_DE 0 A 36
        DEBUT_POUR
        TRACER_SEGMENT (j,0)->(j,cumul[j]/10000)
        TRACER_POINT (j,1/(3*sqrt(2*Math.PI))*exp(-0.5*pow((j-18)/3,2)))
        FIN_POUR
FIN_ALGORITHME
```

```
Xmin: 0 ; Xmax: 36 ; Ymin: 0 ; Ymax: 0.15 ; GradX: 2 ; GradY: 0.01
```