Chapitre 8 : logarithmes

Valentin Melot — Terminale spé maths A

1er mars 2021

1 Logarithme népérien d'un nombre

1.1 Définition

Théorème 1 *Soit $y \in \mathbf{R}_+^*$. Il existe un unique réel $x \in \mathbf{R}$ tel que $y = e^x$.*

L'*existence* de ce réel sera démontrée au chapitre prochain. On peut déjà prouver son *unicité*.

Définition 2 *Soit $y \in \mathbf{R}_+^*$. On note $\ln(y)$, et l'on appelle* logarithme népérien *de y l'unique réel tel que $e^{\ln y} = y$.*

Remarque 3 Les parenthèses peuvent être omises en l'absence de confusion. On peut donc écrire $\ln y$.

Remarque 4 La notation ln est de rigueur en France dans le secondaire. Hors de France et passée la licence, la notation log est souvent préférée ; on la retrouve notamment dans le module `math` de Python. Il est cependant essentiel de bien utiliser la notation ln.

Propriété 5 *1. Pour tout réel strictement positif y, $e^{\ln y} = y$.*

2. Pour tout réel x, $\ln e^x = x$.

Attention, le dernier résultat n'est pas une tautologie !

Propriété 6 *Pour tous $x \in \mathbf{R}$ et $y \in \mathbf{R}_+^*$, on a l'équivalence suivante :*

$$y = e^x \iff x = \ln y.$$

On obtient donc le logarithme par lecture inverse d'un tableau de valeurs de l'exponentielle. On a en particulier :

— $\ln 1 = \dots$;

— $\ln e = \dots$;

— $\ln \frac{1}{e} = \dots$.

Exemple 7 Soit $f : t \mapsto 2e^{-t/2} - 2e^{-t}$ sur $\mathbf{R}_+^*$. Résoudre l'équation $f(t) = 0$, puis donner la valeur du maximum de f.

1.2 Relations fonctionnelles

Les résultats suivant se déduisent immédiatement des règles de calcul de la fonction exponentielle :

Propriété 8 *Soient x et y deux nombres réels strictement positifs, et $n \in \mathbf{N}$. On a :*

1. $\ln(xy) = \ln(x) + \ln(y)$;
2. $\ln \frac{1}{x} = -\ln x$;
3. $\ln \frac{x}{y} = \ln(x) - \ln(y)$;
4. $\ln x^n = n \cdot \ln x$;
5. $\ln x^{-n} = -n \cdot \ln x$.

Exercice 9 En utilisant le premier résultat, exprimer $\ln 1024$ et $\ln 4312$ comme combinaisons linéaires de logarithmes de nombres premiers.

Remarque 10 Cette relation fonctionnelle permettait, avant l'usage des calculatrices, de faciliter les multiplications de grands nombres. Il suffisait en effet de remarquer que pour tous réels strictement positifs x et y,

$$x \times y = \exp(\ln x + \ln y)$$

pour se ramener à une addition, plus simple à effectuer en pratique.

2 Fonction logarithme népérien

2.1 Introduction

Définition 11 *On appelle* fonction logarithme népérien *la fonction :*

$$\begin{array}{rccc} \ln & : & \mathbf{R}_+^* & \longrightarrow & \mathbf{R} \\ & & x & \longmapsto & \ln(x). \end{array}$$

Remarque 12 La fonction logarithme népérien a pour ensemble de définition $]0, +\infty[$. On s'assurera **systématiquement**, avant d'écrire le logarithme d'un nombre, que celui-ci est bien **strictement** positif.

Remarque 13 D'après la propriété 6 :

— Tout nombre réel x admet *au moins* un antécédent par ln dans $\mathbf{R}_+^*$, à savoir e^x. La fonction ln est dite *surjective sur* $\mathbf{R}$.

— Si $x = \ln y = \ln y'$, alors $y = y'$, donc tout nombre réel x admet *au plus* un antécédent par ln. La fonction ln est dite *injective*.

— Donc tout nombre réel x admet *exactement* un antécédent par ln dans $\mathbf{R}_+^*$. On dit que ln est *bijective*.

Propriété 14 *On note* $\mathrm{id}_{\mathbf{R}}$ *et* $\mathrm{id}_{\mathbf{R}_+^*}$ *les fonctions identité (qui associent tout nombre à lui-même) respectivement de* $\mathbf{R}$ *et* $\mathbf{R}_+^*$. *Alors :*

$$\exp \circ \ln = \mathrm{id}_{\mathbf{R}_+^*} \; ; \quad \ln \circ \exp = \mathrm{id}_{\mathbf{R}}.$$

Remarque 15 On dit que ln et exp sont des fonctions *réciproques* l'une de l'autre.

Remarque 16 On remarque que ln et exp transforment la « structure » $(\mathbf{R}, +)$ en la « structure » $(\mathbf{R}_+^*, \times)$:

Ensemble	$\mathbf{R}$		$\mathbf{R}_+^*$
Loi de composition interne	$x + x'$		$y \times y'$
Élément neutre	0		1
Symétrique de x	$-x$		$\frac{1}{y}$
	x	$\overset{\exp}{\longmapsto}$	e^x
	$\ln y$	$\overset{\ln}{\longmapsto}$	y

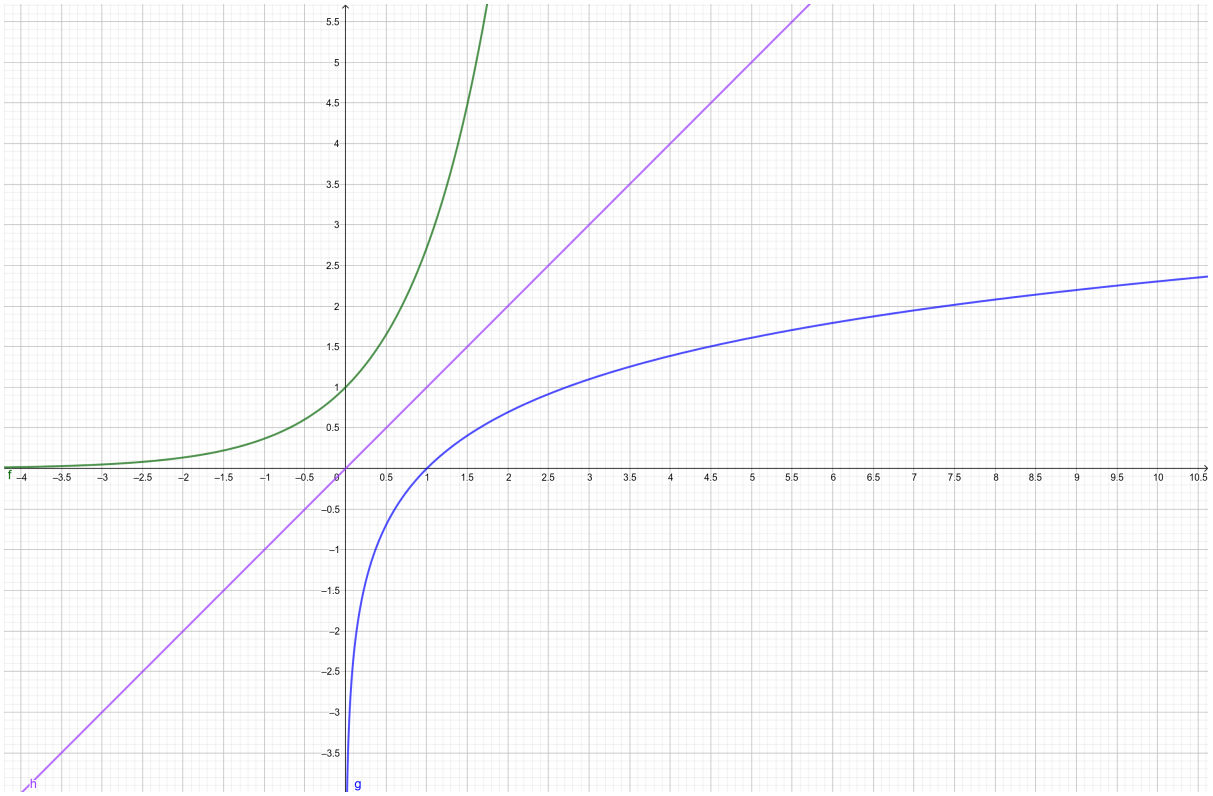
Propriété 17 *Soit $\mathscr{C}$ le graphe de la fonction* exp *et $\mathscr{D}$ le graphe de la fonction* ln. *Soient $(x, y) \in \mathbf{R} \times \mathbf{R}_+^*$. Alors :*

$$(x, y) \in \mathscr{C} \iff (y, x) \in \mathscr{D}.$$

Autrement dit, les courbes $\mathscr{C}$ et $\mathscr{D}$ sont symétriques par rapport à la droite d'équation $y = x$.

Exercice 18 Démontrer que cette propriété est toujours vraie des graphes de deux fonctions réciproques.

On peut donc représenter les courbes des fonctions exp et ln en vis-à-vis, sur un même graphe :



2.2 Limites

Par lecture graphique, on identifie les limites de la fonction ln :

Propriété 19 (limites de référence : logarithme)

$$\lim_{x\to 0^+} \ln x = \dots \ ; \lim_{x\to+\infty} \ln x = \dots .$$

On remarque que ces limites sont « réciproques » de celles de l'exponentielle. On peut les démontrer de deux fonctions :

— Soit en se ramenant à la définition formelle des limites et en utilisant abondamment la propriété 6 ;

— Soit en utilisant la propriété 14, les règles de calcul de limites par composition et les limites de la fonction identité.

Théorème 20 (croissances comparées en 0^+) *Soit $n \in \mathbf{N}$. Alors :*

$$\lim_{x\to 0^+} x^n \ln x = 0.$$

(Preuve exigible pour $n = 1$.)

Théorème 21 (croissances comparées en $+\infty$)

$$\lim_{x\to+\infty} \frac{\ln x}{x^n} = 0.$$

Exercice 22 Soit $n \in \mathbf{N}$. Calculer la limite en 0^+ de $\sqrt[n]{x}\ln(x)$ et la limite en $+\infty$ de $\frac{\ln x}{\sqrt[n]{x}}$.

La fonction ln croît donc vers $+\infty$ plus lentement que toutes les racines. En 0, elle tend vers $-\infty$ plus lentement que les racines tendent vers 0.

2.3 Variations et dérivée

Propriété 23 *La fonction logarithme népérien est strictement croissante sur $\mathbf{R}_+^*$.*

Propriété 24 *Soient $x \in \mathbf{R}$ et $y \in \mathbf{R}_+^*$. On a :*

$$e^x > y \iff x > \ln y.$$

Théorème 25 *La fonction logarithme népérien est dérivable sur $\mathbf{R}_+^*$. Sa dérivée est la fonction inverse.*

On propose une démonstration (exigible) de la valeur de la dérivée en admettant la dérivabilité, et une démonstration (non-exigible) qui ne suppose pas d'admettre ce résultat.

Exercice 26 Donner l'équation de la tangente à la courbe de ln en le point de coordonnées $(1, 0)$.

Conséquence 27

$$\lim_{x\to 0} \frac{\ln(1+x)}{x} = 1.$$

Exercice 28 Tracer le tableau de variations de ln.

Propriété 29 *Soit u une fonction définie sur un intervalle I à valeurs strictement positives. On suppose de plus que u est dérivable sur I. Alors $\ln(u) = \ln \circ u$ est dérivable sur I, et sa dérivée vaut $\frac{u'}{u}$.*

Exercice 30 Étudier les variations de la fonction $x \mapsto x \ln x$.

Exercice 31 Trouver une fonction dont la dérivée est $x \mapsto \frac{1}{x \ln(x)}$.

2.4 Concavité

Propriété 32 *La fonction* ln *est concave sur* $\mathbf{R}_+^*$.

Propriété 33 *Pour tout $x \in \mathbf{R}_+^*$, $\ln(x) \leqslant x - 1$, avec égalité si et seulement si $x = 1$.*

Exercice 34 (inégalité arithmético-géométrique, ultra-classique) Soient x et y deux réels strictement positifs. En utilisant la concavité de ln, démontrer que :

$$\sqrt{xy} \leqslant \frac{x+y}{2}.$$

3 Application : logarithmes et puissances dans d'autres bases

3.1 Puissance d'un nombre

On rappelle que si $n \in \mathbf{N}^*$ et $x \in \mathbf{R}$, alors la racine n-ième de x est l'unique nombre réel positif $\sqrt[n]{x}$ vérifiant :

$$\left(\sqrt[n]{x}\right)^n = x.$$

Cette définition et les propriétés calculatoires des puissances vues au collège justifient que l'on écrive :

$$\sqrt[n]{x} = x^{\frac{1}{n}},$$

et plus généralement, pour tous $p \in \mathbf{Z}$ et $q \in \mathbf{N}^*$,

$$x^{\frac{p}{q}} = \sqrt[q]{x^p} = \left(\sqrt[q]{x}\right)^p.$$

La définition précédente peut donc être étendue aux nombres non-rationnels en posant :

Définition 35 (puissance d'un nombre) *Soit $x \in \mathbf{R}_+^*$. Soit $\alpha \in \mathbf{R}$. On définit x^α comme étant le nombre :*

$$x^\alpha := e^{\alpha \cdot \ln x}.$$

Exercice 36 Démontrer que si $\alpha \in \mathbf{Q}$ et $x \in \mathbf{R}_+^*$, alors les deux définitions précédentes coïncident.

Exercice 37 (ultra-classique) Déterminer la limite de la suite $\left(1+\frac{1}{n}\right)^n$.

Exercice 38 (loi de Poisson) Soit $\alpha \in \mathbf{R}_+^*$. Soit $(p_n)_{n\in\mathbf{N}}$ une suite de réels positifs tels que $\lim\limits_{n\to+\infty} np_n = \alpha$.

1. Démontrer que :
$$\lim_{n\to+\infty} \frac{n\ln(1-p_n)}{-np_n} = 1.$$
2. Soit $k \in \mathbf{N}$. Démontrer que
$$\lim_{n\to+\infty} \frac{n!}{(n-k)!}p_n^k = \alpha^k.$$
3. En déduire le résultat suivant, admis dans le chapitre 5B au sujet de la loi de Poisson :
$$\lim_{n\to+\infty} \binom{k}{n} p_n^k(1-p_n)^{n-k} = \frac{\alpha^k}{k!}e^{-\alpha}.$$

Remarque 39 Cette définition ne permet d'étendre la notation x^α **que si** $x > 0$. En outre, lorsque $x = 0$, si $\alpha > 0$, on peut sans difficulté écrire $0^\alpha = 0$.

Les cas $x < 0$ et $x = 0, \alpha < 0$ en peuvent donner lieu à aucune définition satisfaisante. Le cas $x = \alpha = 0$ est plus ambigu : on ne peut donner de valeur satisfaisante dans tous les contextes, mais on pose parfois comme convention que $0^0 = 1$ pour obtenir des résultats cohérents dans certains domaines.

Définition 40 *Soit $b \in \mathbf{R}_*^+$. On peut définir la fonction* puissance de b *par :*
$$\begin{array}{rccc} p_b & : & \mathbf{R} & \longrightarrow & \mathbf{R}_+^* \\ & & x & \longmapsto & b^x = e^{x\ln b}. \end{array}$$

La notation p_b n'est pas standard. L'étude de cette fonction n'est pas très intéressante, puisqu'il s'agit simplement de la composée de l'exponentielle avec une fonction linéaire.

Exercice 41 Étudier la fonction p_b : calcul des dérivées première et seconde, variations, convexité et limites.

3.2 Logarithmes dans d'autres bases

Définition 42 *Soit $b \in]1,+\infty[$. On définit pour $x \in \mathbf{R}$, le* logarithme de x en base b par *:*
$$\log_b(x) = \frac{\ln x}{\ln b}.$$

Propriété 43 *Soit $b > 1$. Pour tout $x \in \mathbf{R}$ et $y \in \mathbf{R}_+^*$, on a l'équivalence suivante :*
$$y = b^x \iff x = \log_b(y).$$

La *fonction logarithme en base b* est la fonction $x \longmapsto \log_b(x)$, définie sur $\mathbf{R}_+^*$.

Exercice 44 Soit $b > 1$. Démontrer que $\log_b$ vérifie les mêmes relations fonctionnelles que ln.

Exercice 45 Soit $b > 1$. Démontrer que $\log_b$ est la fonction réciproque de p_x.

Exercice 46 Soit $b > 1$. Étudier la fonction $\log_b$ (dérivées première et seconde, variation, convexité).

Les fonctions les plus utilisées sont :

— $\log_e = \ln$, que l'on appelle *logarithme naturel*,

— $\log_{10}$, le *logarithme décimal*, plus simplement noté log,

— $\log_2 = \text{lb}$, le *logarithme binaire*.

On utilise fréquemment le logarithme décimal pour comparer le niveau relatif de deux grandeurs, en décibels. Par exemple, si Y_1 est la puissance d'un signal après transmission et Y_0 la puissance du signal initial, l'amplification, exprimée en décibels, est :

$$10 \times \log_{10} \frac{Y_1}{Y_0}.$$

Ainsi, un signal atténué de 20 dB est un signal dont la puissance a été divisée par 100.

Il est utile de savoir que $\log 2 \approx 0,3$, $\log 3 \approx 0,5$ et $\log 5 \approx 0,7$. Un son amplifié de 3 dB correspond à une puissance sonore doublée.

On a par ailleurs $\text{lb}\, 1000 \approx 10$, donc $\text{lb}\, 10 \approx 3,3$.

3.3 Approfondissement : vers la théorie de l'information

La *théorie de l'information* est un domaine au croisement de l'informatique et des mathématiques. Il repose sur le fait d'exprimer la quantité d'information dont un message est porteur.

Considérons un ordinateur classique. Ses informations sont stockées sur des *bits* (*binary units*), c'est-à-dire des unités élémentaires ne pouvant prendre que la valeur `0` ou `1`.

— Avec un bit, on ne peut stocker que deux informations différentes, codées par `0` ou `1`.

— Avec deux bits, on peut stocker quatre informations différentes : `00`, `01`, `10`, `11`.

— Avec trois bits, huit informations différentes : `000`, `001`, `010`, `011`, `100`, `101`, `110` et `111`.

— Et ainsi de suite.

Plus généralement, n bits permettent de stocker 2^n valeurs distinctes pour une information.

Et inversement, de combien de bits aurait-on besoin pour stocker une information dont on connaît le nombre de valeurs différentes susceptibles d'être prises ? Si l'on souhaite stocker l'identification d'un élève dans une classe de 31 personnes, on peut envisager différentes façons :

— Stocker une chaîne de caractères comportant le prénom et le nom. Dans les représentations les plus simples, un caractère est stocké sur huit bits. Si l'élève ayant le prénom et le nom le plus long requiert vingt caractères, alors il faudra $20 \times 8 = 160$ bits.

— Mais s'il n'y a pas deux élèves portant le même nom de famille, cette information suffit. On pourrait par exemple se contenter de huit caractères, quitte à tronquer les noms : 64 bits suffiront.

— D'après les spécialistes, il n'y aurait que 350 000 noms différents dans la population française. Or,

$$\operatorname{lb} 360\,000 \approx 18.46.$$

Autrement dit, en numérotant les noms qui existent, avec seulement 19 bits, on peut stocker l'information de chaque nom de famille.

— Et finalement, sachant que la liste des élèves est fixe, puisque $\operatorname{lb} 31 \approx 4.95$, seuls 5 bits suffisent.

Autrement dit, en analysant mieux le problème pour exploiter au maximum les données du contexte, on constate qu'une information que l'on aurait naïvement codée sur 160 bits peut en réalité se coder sur seulement 5. Plus généralement, si n valeurs différentes sont possibles, alors $\operatorname{lb} n$ bits suffisent à représenter cette information.

On peut trouver de très nombreuses applications à ce genre de considérations.

Transmissions. On peut limiter les erreurs de transmission. Dans l'exemple précédent, si une personne envoie le nom d'un élève représenté sous la première forme et commet une erreur (une lettre modifiée), il sera facile de la corriger à partir des autres lettres. On peut, à l'aide de la théorie de l'information, estimer à quel point un signal peut être « dégradé » par des erreurs tout en restant lisible.

Compression. On peut définir des algorithmes de compression efficaces, qui permettent d'économiser de l'espace de stockage sans perdre d'information. Par exemple, en français, en tenant compte des conjugaisons et accords, on peut trouver moins de $N =$ deux millions de mots. Une phrase de k mots peut donc être codée sur :

$$\operatorname{lb} N^k = k \cdot \operatorname{lb} N \approx 21 \times k \text{ bits.}$$

À comparer avec la représentation « naïve », qui requiert huit bits par caractère. On pourrait faire encore mieux en exploitant les différences de fréquences des mots.

Mots de passe. On peut mesurer la forme d'un mot de passe. Un mot de passe est une succession de « caractères » d'un alphabet déterminé. Par exemple, `03485` est un mot de passe de l'alphabet des chiffres (cinq caractères, dix possibilités pour chacun), `TPLV984` un mot de passe de l'alphabet majuscules + chiffres (sept caractères, trente-six possibilités pour chacun), `bavard oiseau café` un mot de passe de l'alphabet des mots de la langue française (trois caractères, deux millions de possibilité pour chacun). La force d'un mot de passe est le nombre de bits nécessaires pour le stocker avec les informations dont on dispose :

— `03485` correspond à une force de $5 \times \text{lb}\, 10$, soit 16 bits.

— `TPLV984` à une force de $7 \times \text{lb}\, 36$, soit 36 bits.

— `bavard oiseau café` à une force de $3 \times \text{lb}\, 2\,000\,000$, soit 62 bits [1].

Le dernier est donc le plus robuste. De façon générale, un mot de passe consistant en une série de mots est à la fois plus robuste et plus facile à retenir qu'un mot de passe consistant en une suite de caractères alphanumériques aléatoires. Comparer par exemple `Kiwi cosmétique archimédéen mangeant du cheval givré` et `lc|//Z2lal*X`.

L'agence nationale de la sécurité des systèmes d'information (ANSSI) estime qu'un mot de passe peut être considéré comme « faible » à partir de 64 bits, « moyen » à partir de 80, et « fort » à partir de 100. Des forces minimales sont définies pour chaque catégorie d'usage (on n'exige pas la même force de mot de passe pour protéger un ordinateur du réseau du lycée, un compte personnel sur un réseau social ou une base de données sensible). Un mot de passe de moins de 64 bits ne présente pas la moindre garantie de sécurité.

Problème ouvert 47 Comment pourrait-on estimer la force d'un mot de passe dont on sait qu'il correspond à une phrase en français ?

4 Calculs de logarithmes en pratique

4.1 Bibliothèque de Python

La bibliothèque standard `math` de Python comporte une fonction `log`. Attention, celle-ci est nommée selon les conventions anglo-saxonnes : elle correspond au logarithme naturel, et non pas décimal. Un second paramètre, facultatif, permet de définir une base. Par exemple :

```
>>> from math import log, exp
>>> e = exp(1)
>>> log(e)
1.0
>>> log(1/e)
-1.0
>>> log(exp(3))
3.0
>>> log(10)
2.302585092994046
>>> 10**2.3026
200.72432216432944
>>> exp(2.3026)
10.000149071170643
>>> log(10,10)
1.0
>>> log(2,10)
0.30102999566398114
>>> 10**(log(2,10)) #Noter l'erreur d'arrondi
1.9999999999999998
>>> log(10,2)
3.3219280948873626
>>> log(1024,2)
10.0
```

1. En réalité moins que cela, notamment pour des raisons statistiques (les mots utilisés ici sont des mots fréquents, donc qui confèrent moins de force au mot de passe).

4.2 Complément : algorithmes de Briggs

Le mathématicien John Napier (1550–1617) est l'auteur de la première table de logarithmes décimaux. Henry Briggs (1556–1630) découvrit, peu après lui, des méthodes algorithmiques pour préciser grandement ces calculs, que nous allons voir ci-après.

4.2.1 Première méthode : en se fondant sur le nombre de chiffres

Le constat de Briggs est le suivant : si le nombre n, écrit en base 10, a k chiffres, alors $10^{k-1} \leqslant n \leqslant 10^k$. Donc $k-1 \leqslant \log n \leqslant k$.

Il suffit alors d'élever un nombre à une certaine puissance et de compter ses chiffres pour avoir son logarithme à une certaine précision. En effet, si n^m a k chiffres, alors $k-1 \leqslant \log(n^m) \leqslant k$, d'où

$$\frac{k-1}{m} \leqslant \log n \leqslant \frac{k}{m}.$$

Prendre une puissance suffisamment grande permet de déterminer le logarithme recherché. Briggs trouve un algorithme permettant d'élever des nombres à la puissance 10^14 en procédant ainsi :

— Calculer x^2, $x^4 = (x^2)^2$, $x^8 = (x^4)^2$, $x^{10} = x^2 \times x^8$;

— Calculer $x^{20} = (x^{10})^2$, $x^{40} = (x^{20})^2$, $x^{80} = (x^{40})^2$; $x^{100} = x^{20} \times x^{80}$;

— Calculer $x^{200} = (x^{100})^2$, $x^{400} = (x^{200})^2$, $x^{800} = (x^{400})^2$; $x^{1000} = x^{200} \times x^{800}$;

— Etc.

En seulement **soixante multiplications**, on calcule $x^{10^{15}}$. En comptant le nombre de chiffres de ce nombre, on en déduit la valeur de $\log x$ avec une précision de **quatorze chiffres après la virgule**.

En fait, cet algorithme n'est pas applicable tel quel en pratique. Pour $x = 1$, par exemple, 2^{10^3} est déjà un nombre à 300 chiffres. Briggs utilisait plutôt des astuces pour déterminer, étape après étape, combien de chiffres avaient les produits, en ne calculant explicitement qu'une partie du résultat.

En Python, si `n` est une variable entière, alors `len(str(n))` permet de connaître le nombre de chiffres de `n` en base 10. On propose donc la fonction suivante :

```
def briggs(x, precision):
    """Renvoie la valeur de log(‘x‘), avec une précision de
    ‘precision‘ chiffres après la virgule, selon la métode de
    Briggs."""
    #Première étape : calcul de x^(10^precision) par quatraines
    #Invariant : avant le i-ième tour de boucle, on a :
    #p10 = x^(10^i)
    p10 = x
    for i in range(precision+1):
        p2 = p10*p10
        p4 = p2*p2
        p8 = p4*p4
        p10 = p2*p8
    #À ce stade, p10 = x^(10^(precision+1))
    #Comptage des chiffres
    nb_chiffres = len(str(p10))
    return nb_chiffres / 10**(precision+1)
```

On peut l'exécuter, pour voir que :

```
>>> briggs(2, 3)
0.3011
>>> briggs(2, 4)
0.30103
>>> briggs(2, 5)
0.30103
```

Cependant, la manipulation de nombres aussi gros pose toutefois aussi problème aux ordinateurs, et il n'est pas possible en pratique de dépasser `briggs(2,5)` — ce qui revient déjà à travailler avec des nombres à 300 000 chiffres !

4.2.2 Deuxième méthode : avec des suites adjacentes

Briggs propose une seconde méthode pour calculer le logarithme décimal de x. Il construit pour cela deux suites $(u_n)_{n\in\mathbf{N}}$ et $(v_n)_{n\in\mathbf{N}}$ qui encadrent le nombre x et convergent vers celui-ci. Il construit dans le même temps les suites $(\log u_n)_{n\in\mathbf{N}}$ et $(\log v_n)_{n\in\mathbf{N}}$, qui correspondent aux logarithmes décimaux des deux précédentes. Puisque log est une fonction croissante, ces suites encadrent donc également $\log x$, et convergeront ensemble vers sa valeur.

Autrement dit, on construit ces suites de façon à garantir le respect simultané des conditions suivantes, pour tout $n \in \mathbf{N}$:

$$u_n \leqslant u_{n+1} \leqslant x \leqslant v_{n+1} \leqslant v_n \ ;$$

$$\log u_n \leqslant \log u_{n+1} \leqslant \log x \leqslant \log v_{n+1} \leqslant \log v_n.$$

L'idée est alors de choisir des suites u_n et v_n dont on peut calculer les logarithmes en utilisant les relations fonctionnelles. Plus précisément, on utilise le fait que :

$$\log \sqrt{u_n v_n} = \frac{\log u_n + \log v_n}{2}.$$

Pour construire u_{n+1} et v_{n+1}, on procède par la méthode de *dichotomie*. On calcule la moyenne géométrique $\alpha = \sqrt{u_n v_n}$, et :

— Si $\alpha \leqslant x$, alors on pose $u_{n+1} = \alpha$ et $v_{n+1} = v_n$.

— Si $\alpha \geqslant x$, alors on pose $u_{n+1} = u_n$ et $v_{n+1} = \alpha$.

On met à jour les valeurs connues de $\log u_{n+1}$ et $\log v_{n+1}$, connues, en conséquence. On pourra alors s'arrêter lorsque ces deux valeurs seront suffisamment proches, et on connaîtra une majoration sur l'erreur.

Pour l'initialisation, il suffit de choisir u_0 et v_0 dont on peut calculer le logarithme et qui respectent la condition initiale d'encadrement. Par exemple, si $1 \leqslant x \leqslant 10$, alors $u_0 = 1$ et $v_0 = 10$ conviennent parfaitement.

Exercice 48 Pourquoi suffit-il de savoir calculer le logarithme de n'importe quel nombre entre 1 et 10 pour pouvoir calculer le logarithme de tout réel strictement positif ?

Exercice 49 Démontrer que la suite $(\log v_n - \log u_n)_{n\in\mathbf{N}}$ est une suite géométrique, et donner sa raison. En déduire sa limite.

Exercice 50 (théorème des suites adjacentes) Démontrer que si $(a_n)_{n\in\mathbf{N}}$ et $(b_n)_{n\in\mathbf{N}}$ sont deux suites telles que $(a_n)_{n\in\mathbf{N}}$ est croissante, $(b_n)_{n\in\mathbf{N}}$ est décroissante et $\lim\limits_{n\to+\infty} b_n - a_n = 0$, alors ces deux suites convergent vers une limite commune.

Exercice 51 Déduire des deux résultats précédents que $(\log u_n)_{n\in\mathbf{N}}$ et $(\log v_n)_{n\in\mathbf{N}}$ convergent vers $\log x$.

On propose, en conséquence, l'implémentation suivante :

```
def logdichot(x, precision):
    """Renvoie la valeur de log(‘x‘), calculée par dichotomie, avec une
    erreur au maximum égale à ‘precision‘. ‘x‘ est un nombre compris entre
    1 et 10, et ‘precision‘ est un nombre réel strictement positif
    arbitrairement proche de 0."""
    from math import sqrt
    u = 1
    v = 10
    logu = 0
    logv = 1
    delta = 1
    while delta > precision:
        alpha = sqrt(u*v)
        logalpha = (logu + logv)/2
        if alpha >= x:
            v = alpha
            logv = logalpha
        else:
            u = alpha
            logu = logalpha
        delta /= 2
    return (logu+logv)/2
```

L'algorithme permet d'obtenir très rapidement des résultats satisfaisants :

```
>>> logdichot(2,0.01)
0.30078125

>>> logdichot(2,0.000001)
0.30102968215942383

>>> logdichot(2,0.0000000000001)
0.30102999566398125
```

Exercice 52 Exécuter *à la main* un appel de `logdichot(2, 0.05)`.

Exercice 53 Adapter l'algorithme pour qu'il puisse calculer le logarithme en n'importe quelle base b d'un nombre compris entre 1 et b.

Exercice 54 Modifier l'algorithme pour qu'il affiche le nombre de tours de boucle. Conjecturer une relation explicite entre `precision` et ce nombre de tours. Cette relation fera intervenir la fonction partie entière, et le logarithme binaire de `precision`.

Exercice 55 En conséquence, adapter l'algorithme pour qu'il commence par calculer le nombre de tours à faire, puis comporte une boucle `for` à la place da la boucle `while`. Cette modification permet de limiter le nombre de comparaisons de nombres décimaux.

Exercice 56 Modifier l'algorithme précédent pour qu'il fonctionne par « trichotomie » plutôt que par dichotomie, c'est-à-dire pour qu'il choisisse u_{n+1} et v_{n+1} parmi u_n, $\sqrt[3]{u_n^2 v_n}$, $\sqrt[3]{u_n v_n^2}$ et v_n.

Cette nouvelle stratégie permet-elle d'économiser des tours de boucles ? Permet-elle de faire moins de calculs ?

Les acquis de ce chapitre

Démonstrations exigibles :

1. Calcul de la dérivée de ln, en admettant qu'elle est dérivable.
2. Calcul de la limite de $x \ln x$ en 0.

Savoirs et savoirs-faire indispensables.

1. Définition du logarithme népérien comme antécédent par la fonction exponentielle.
2. Relations fonctionnelles vérifiées par ln
3. Tracé de la courbe représentative de ln, lien avec celle d'exp.
4. Limites de ln, croissance comparée avec les polynômes.
5. Dérivée, variations, concavité de ln.
6. Définition et propriétés élémentaires du logarithme en base b.
7. Définition de la puissance d'un nombre.

Démonstrations qu'il faut avoir comprises :

— Cas généraux des théorèmes de croissance comparée de ln.

— Calcul de la limite de $\dfrac{\ln(1+x)}{x}$ en 0.

— Premier algorithme de Briggs.

Principaux approfondissements.

1. Éléments de théorie de l'information.
2. Deuxième algorithme de Briggs (par dichotomie).