FLUORESCIENCES

Analyse

Ana

Jean-Romain Heu

Loïc Teyssier

DUNOD

Le pictogramme qui figure ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du photocopillage.
Le Code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.
Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 20, rue des Grands-Augustins, 75006 Paris).

© Dunod, 2022

11, rue Paul Bert, 92240 Malakoff
www.dunod.com

ISBN 978-2-10-082159-4

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2° et 3° a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

Table des matières

Se repérer dans le livre VI
Avant-propos VIII

CHAPITRE 1 **UNE HISTOIRE LAPIDAIRE DU CALCUL** 2
1 Des cailloux aux ordinateurs 4
2 Calculer avec des nombres et des symboles 9
3 Représenter et calculer avec des nombres 12

CHAPITRE 2 **LES NOMBRES RÉELS** 22
1 Introduction 24
2 Une construction des réels 26
3 Les parties entières 33
4 Les bornes supérieures et inférieures 34
5 Les intervalles de la droite réelle achevée 39
6 Application Python : coder les rationnels 40
Ce qu'il faut retenir 42
Exercices 43

CHAPITRE 3 **FORMULES ET ÉVALUATION** 44
1 Les fonctions élémentaires 46
2 Les formules et les évaluateurs 53
3 Les fonctions usuelles 60
4 Liste des formules standard 64
5 Application Python : manipuler et évaluer des formules 65
Ce qu'il faut retenir 70
Exercices 71

CHAPITRE 4 **LES SUITES NUMÉRIQUES CONVERGENTES** 72
1 Les suites numériques 76
2 La convergence d'une suite 78
3 Les opérations sur les limites 83
4 Propriétés de la limite liées à l'ordre sur $\mathbb{R}$ 86

5 Comparaison d'ordres de grandeur classiques ... 91
6 Pour aller plus loin : suite logistique et chaos ... 92
Ce qu'il faut retenir ... 96
Exercices ... 97

CHAPITRE 5 LES FONCTIONS CONTINUES SUR UN INTERVALLE ... 98

1 Introduction ... 100
2 Limite d'une fonction réelle en un point ... 101
3 Ordre et limites ... 106
4 La continuité d'une fonction ... 108
5 Les grands théorèmes de l'analyse des fonctions continues ... 112
6 Pour aller plus loin : les suites définies par itération ... 115
Ce qu'il faut retenir ... 118
Exercices ... 119

CHAPITRE 6 LES FONCTIONS DÉRIVABLES ... 120

1 Dérivée en un point ... 122
2 Propriétés algébriques de la dérivation ... 124
3 Les dérivées d'ordre supérieur et les classes de dérivabilité ... 125
4 Les théorèmes fondamentaux ... 127
5 Pour aller plus loin : perceptron et descente de gradient ... 132
Ce qu'il faut retenir ... 134
Exercices ... 135

CHAPITRE 7 LES FONCTIONS USUELLES : CONSTRUCTION, ÉTUDE ET TRACÉ ... 136

1 Une construction de l'exponentielle ... 138
2 Les réciproques des fonctions standards ... 145
3 Les dérivées symboliques ... 147
4 Propriétés fondamentales des fonctions usuelles ... 152
5 Plan d'étude d'une fonction usuelle ... 157
6 Développements limités et évaluation numérique de fonctions usuelles ... 158
Ce qu'il faut retenir ... 163
Exercices ... 164

CHAPITRE 8 **LE CALCUL INTÉGRAL** ... 166
1 Construction de l'intégrale ... 168
2 Propriétés de l'intégrale ... 171
3 Le théorème fondamental du calcul différentiel ... 172
4 Les primitives symboliques ... 174
5 Pour aller plus loin : intégration numérique ... 183
Ce qu'il faut retenir ... 184
Exercices ... 185

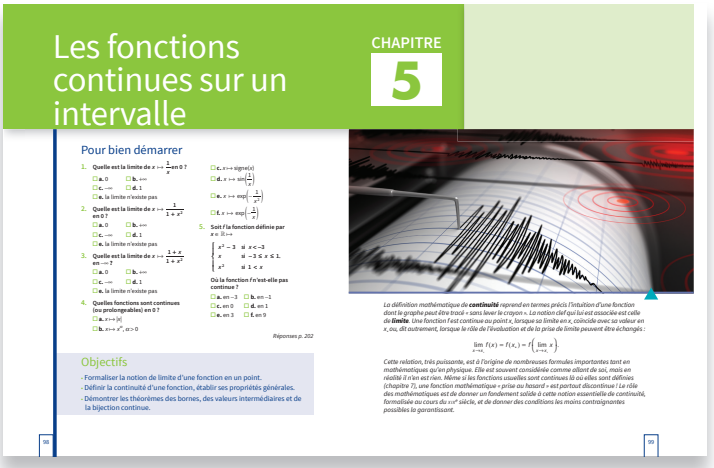
CHAPITRE 9 **LES ÉQUATIONS DIFFÉRENTIELLES** ... 186
1 Généralités ... 188
2 Équations différentielles résolubles ... 191
3 Résoudre une équation différentielle avec PYTHON ... 199
Ce qu'il faut retenir ... 202
Exercices ... 203

Corrigés ... 204
Index ... 207
Crédits iconographiques ... 208

Se repérer dans le livre

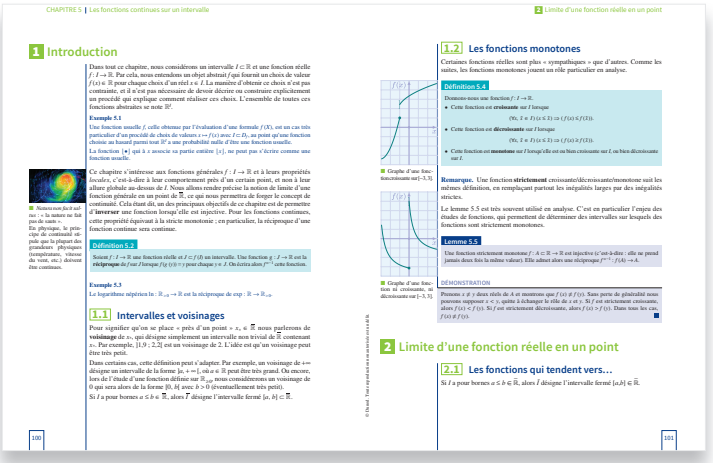
Ouverture de chapitre

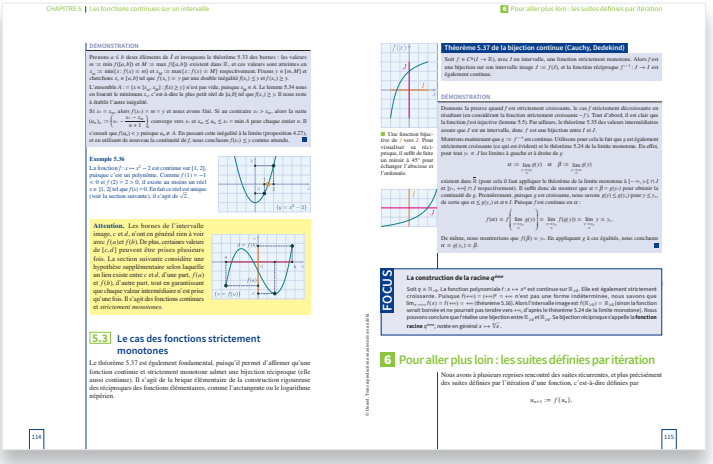
- QCM pour se tester sur les prérequis de Terminale.
- Un exemple concret pour introduire le sujet du chapitre.
- Ce que l'on maîtrisera à la fin du chapitre.



Le cours

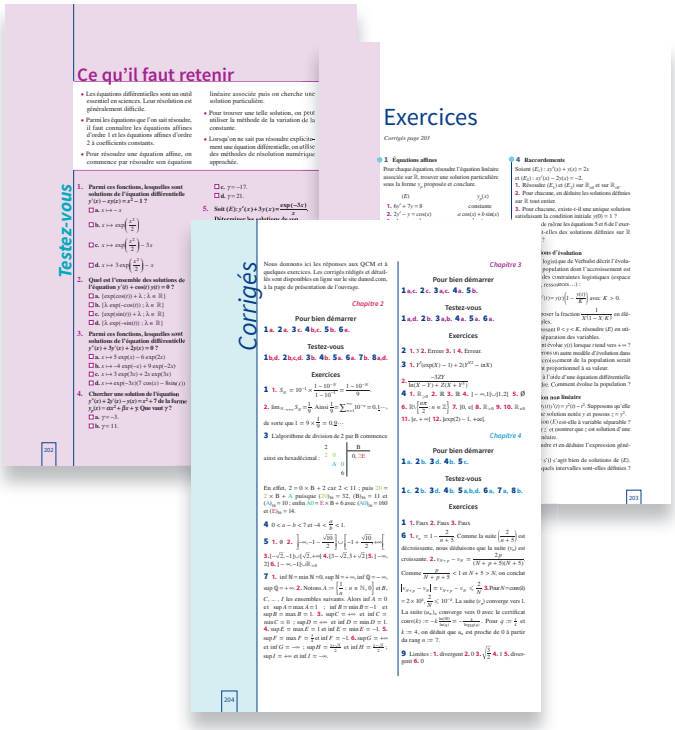
- Le **cours** est illustré par des figures et de nombreux exemples.
- Les **focus** développent un sujet de recherche, une application, un thème d'actualité.
- Des repères historiques.
- Des conseils et erreurs à éviter.





En fin de chapitre

- Un **résumé** de ce qu'il faut retenir.
- Les **QCM** et **exercices** permettent de vérifier ses connaissances et de s'entraîner aux examens.
- Les **corrigés** sont détaillés à la fin du livre.



En fin d'ouvrage

- Un **index** pour retrouver rapidement les notions principales.

Les + en ligne

Retrouvez sur la page dédiée à l'ouvrage sur le site dunod.com :

- Des compléments de cours.
- Les corrigés détaillés de tous les exercices.

Avant-propos

La place fondamentale des mathématiques dans l'enseignement supérieur n'est pas discutée. Au contraire, elles sont à la fois le langage des disciplines scientifiques et techniques, permettant ainsi la description théorique et la modélisation, ainsi que leur moyen d'action, en permettant la conception et la prévision par le calcul. Le présent ouvrage est axé sur l'**analyse mathématique**, avec un point de vue algorithmique permettant d'éclairer la théorie et d'exposer les difficultés de la pratique. Si cette discipline est fondamentale pour les applications, elle présente néanmoins une réelle difficulté pour les étudiants. Il nous a immédiatement paru important d'introduire en parallèle le **calcul numérique** et le **calcul formel**. Expliquons brièvement ces deux termes, puisque cette dichotomie constitue le cœur de l'ouvrage.

1. L'architecture matérielle des ordinateurs permet d'effectuer nativement des opérations arithmétiques (sur les entiers, les flottants...) : cette fonctionnalité fonde l'analyse numérique, c'est-à-dire le calcul approché de valeurs. Le problème majeur associé est celui de la qualité de l'approximation (certification des chiffres significatifs, propagation des erreurs, etc.), qui rejoint des préoccupations relatives aux sciences expérimentales.
2. Par ailleurs, des logiciels plus sophistiqués permettent de manipuler des expressions symboliques (fractions d'entiers, formules trigonométriques...) et de calculer formellement à l'aide de ces objets. Ces fonctionnalités sont de plus en plus présentes sur les calculatrices utilisées dans le secondaire, mais elles sont probablement peu ou mal exploitées. Le problème majeur associé est celui de l'impossibilité de trouver le résultat dans la classe des objets symboliques, qui est un problème rencontré aussi dans les autres sciences (formules « explicites », « analytiques » ou encore « sous forme close »).

La meilleure illustration de cette dualité est la fonction de répartition gaussienne, omniprésente en sciences : la *théorie* construit la primitive $F(X) := \int_{-\infty}^{X} \exp(-x^2)\,\mathrm{d}x$, le *calcul formel* permet d'affirmer que $F(X)$ n'est pas une formule « explicite » et donc nous contraint à effectuer un *calcul numérique* approché des valeurs, par exemple pour constituer des tables précalculées destinées aux utilisateurs.

Notre propos n'est pas tant de fournir un manuel de techniques élaborées, d'analyse numérique ou de calcul formel, même si nous souhaitons offrir une introduction aux grands problèmes que ces deux approches complémentaires permettent de formuler. Les méthodes naïves de résolution que nous proposons ont pour rôle d'illustrer les fondements théoriques de l'analyse réelle et de souligner la transcendance intrinsèque du monde des nombres réels.

Une fois qu'ils auront assimilés ces concepts et algorithmes basiques, les étudiants seront mieux armés pour appréhender les algorithmes optimaux et les logiciels (MatLab, SciLab, Mathematica, Maple, SageMath...) présents dans l'industrie ou les laboratoires de recherche.

Contenu

Avant l'apparition et l'utilisation intensive des calculateurs articiels – il n'y a guère plus que 70 ans – la production de livres de tables de valeurs numériques était un enjeu majeur : cet artisanat mathématique, facilité par l'introduction de l'écriture

décimale des réels à la fin de la Renaissance, a accompagné les diverses révolutions technologiques et industrielles du XIX^{e} siècle et de la première moitié du XX^{e} siècle, en motivant d'ailleurs le financement et la conception des premiers calculateurs automatiques afin d'améliorer la qualité des calculs. Ainsi, le premier chapitre du présent ouvrage est consacré à l'**histoire du calcul** et des techniques.

Une fois présentée la **construction des nombres réels** par les développements décimaux et l'introduction du concept de **limite**, le but principal du discours est de démontrer et de donner une incarnation concrète aux **grands théorèmes de l'analyse réelle** (valeurs intermédiaires, accroissements finis, équivalence primitive/intégrale), en se focalisant sur une approche constructive. Ces chapitres présentent notamment une construction de l'**intégrale** des fonctions de classe C^1 et décrivent l'espace des solutions d'**équations différentielles** simples.

Les thèmes abordés dans ce livre sont donc en grande partie classiques ; nous avons toutefois choisi de mettre en avant la classe des **fonctions usuelles**. Ces fonctions – dont les étudiants sont si familiers qu'ils en oublient facilement que d'autres fonctions existent – occupent une place particulière dans la discipline, car elles sont la charnière entre calcul symbolique et calcul numérique : ce sont les fonctions obtenues par l'évaluation d'une formule sur son ensemble de définition. Une des tâches du chapitre qui leur est consacré est de construire (et de dériver) l'**exponentielle réelle** et les **fonctions circulaires**, ce qui débouchera finalement sur la construction des **fonctions réciproques** standards. Nous y démontrons également les **théorèmes de continuité et de dérivabilité** usuels, qui donnent un cadre pratique très concret pour discuter de la continuité et de la dérivabilité des fonctions usuelles.

À l'issue de l'ouvrage, le lecteur studieux saura écrire des algorithmes permettant de calculer des valeurs approchées de fonctions usuelles (par exemple, e^{π}) avec une précision certifiée, mais aussi des solutions exactes ou approchées d'équations différentielles et d'intégrales.

Public visé et prérequis

L'ouvrage s'adresse à des étudiants ayant une connaissance au moins élémentaire en mathématiques (ayant choisi la spécialité « mathématiques » ou l'option « mathématiques complémentaires » du nouveau baccalauréat général) et ayant déjà pratiqué les bases du langage PYTHON (normalement à travers la spécialité « mathématiques » en Première), se destinant à des études supérieures scientifiques principalement en mathématiques ou en informatique, mais également dans toute direction utilisant du calcul scientifique et nécessitant une bonne compréhension de l'analyse mathématique (physique théorique ou appliquée, chimie, géosciences...).

Il est conseillé au lecteur de se familiariser avec le langage de programmation PYTHON, ce livre n'étant pas un ouvrage de référence sur ce langage (nous en utiliserons la version 3). Ce langage est devenu omniprésent, de la fin du secondaire au monde de la recherche et de l'industrie : PYTHON permet d'effectuer du calcul numérique efficace (grâce à la bibliothèque `numpy`) et symbolique (grâce à la bibliothèque `sympy`).

Cet ouvrage ne couvre pas non plus les fondements mathématiques du raisonnement, de l'algèbre, de la théorie des ensembles ou de l'arithmétique. Nous renvoyons le lecteur à l'excellent ouvrage *Mathématiques* dans la collection « Fluorescences », publié chez Dunod, pour un panorama plus exhaustif de ces notions basiques.

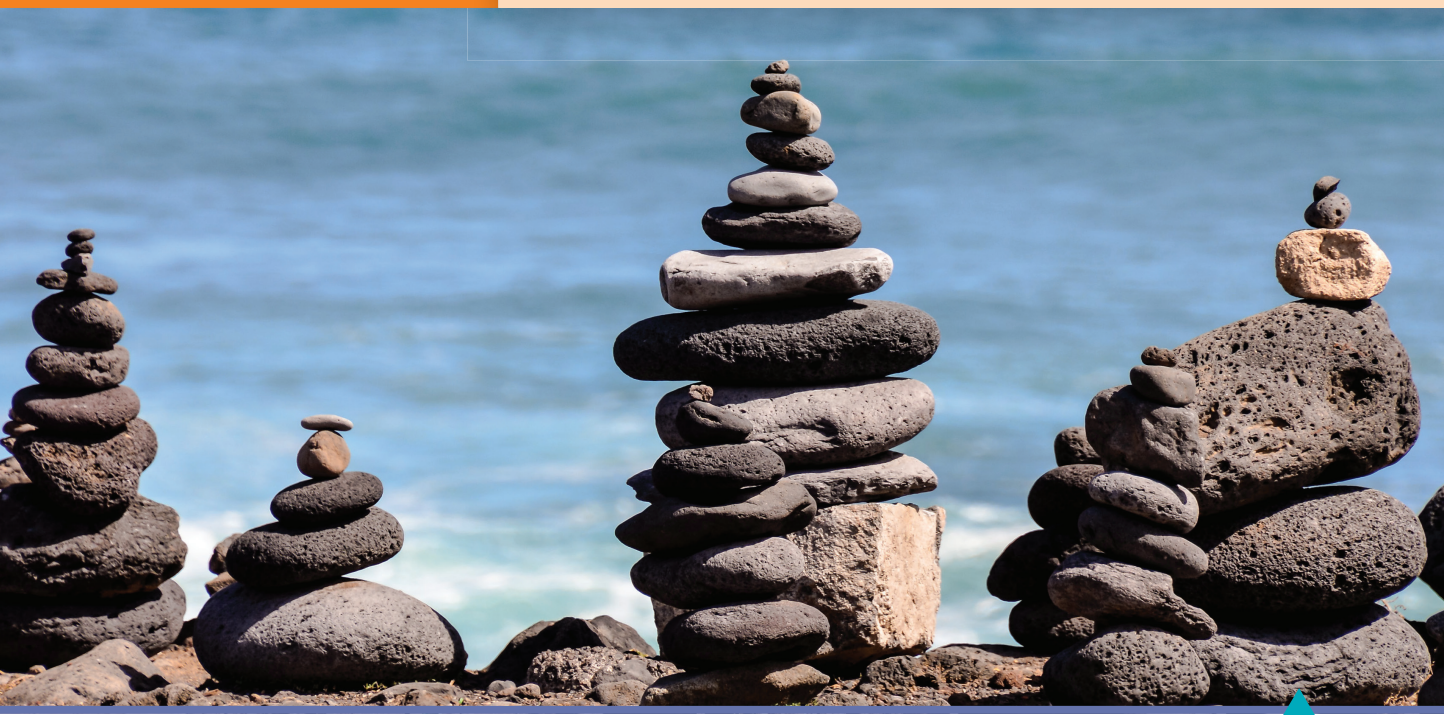
Une histoire lapidaire du calcul

$\simeq -3000$	Mésopotamie, écriture cunéiforme, représentation des nombres en base 60
$\simeq -2000$	Mésopotamie, tablettes scolaires d'apprentissage des quatre opérations, racines carrées et cubiques
$\simeq -1000 \rightarrow 0$	Chine, numération décimale positionnelle, utilisation d'abaques et premiers algorithmes
$\simeq 600$	Inde, 0 devient un nombre, découverte des nombres négatifs, règle des signes et perfectionnement des pratiques opératoires
$800 \rightarrow 1500$	Empire arabe, introduction de l'algèbre et des polynômes, diffusion du système décimal indien, premières tables de calcul trigonométrique
XVI^{e} siècle	Europe, utilisation des nombres complexes, représentation décimale « à virgule » des nombres, calcul symbolique
XVII^{e} siècle	Europe, logarithmes, découverte de la dérivée
XVIII^{e} siècle	Europe, exponentielle, développements limités, équations différentielles
XIX^{e} siècle	Europe, théories des nombres réels, fondement de l'analyse mathématique
XX^{e} siècle	Théorie de la calculabilité, naissance de l'informatique

Objectifs

- Discuter la nature des nombres et leur représentation dans un ordinateur.
- Définir, grâce aux machines de Turing, la notion de procédé calculable.
- Comprendre la place du calcul dans l'histoire des sciences et des techniques.
- Distinguer deux types de calculs : symbolique et numérique.

CHAPITRE 1



La nécessité d'effectuer des additions pour gérer des cités-États grandissantes a forcé l'émergence de l'écriture positionnelle des nombres dès l'Antiquité babylonienne. L'architecture, l'astronomie, la balistique antiques puis médiévales demandent de plus en plus de calculs. Les opérations financières de la Renaissance mènent à la découverte des logarithmes : transformant multiplications en additions, ils permettent aux humains de faire plus facilement plus d'opérations. Toujours plus de calculs pour prévoir l'évolution des systèmes gouvernés par les lois de la physique, pour construire les machines des révolutions industrielles initiées au XIX^e siècle, pour envoyer des humains sur la Lune ou sur Mars. Pour savoir s'il a plus hier, ou streamer la Reine des neiges *dans sa location de tourisme en Creuse. Pour visiter un supermarché en réalité virtuelle dans un* metaverse, *ou simuler l'explosion d'une bombe atomique. Notre société s'ancre de plus en plus dans une réalité calculée. Mais qui fait les calculs ? Et comment ?*

1 Des cailloux aux ordinateurs

Ada Lovelace (1815–1852)
Mathématicienne et inventrice anglaise, considérée comme la pionnière de l'informatique. Elle est l'auteure du premier programme informatique.

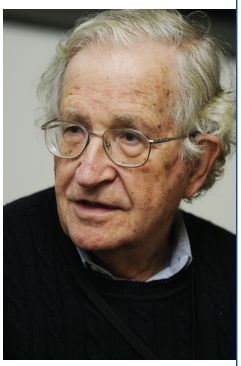
« La machine analytique n'a nullement la prétention de créer quelque chose par elle-même. Elle peut exécuter tout ce que nous saurons lui ordonner d'exécuter [. . .] Son rôle est de nous aider à effectuer ce que nous savons déjà dominer. »

Lovelace, parlant de la machine analytique de Babbage (section 1.4).

1.1 Un mot sur les langages

L'humain est un animal singulier, probablement moins par son sens de l'humour que par sa capacité à transmettre des informations immatérielles à sa descendance. L'accumulation de ce savoir, et sa conservation au-delà de la vie des individus, caractérise l'humain comme animal social, certes, mais surtout culturel. Le langage est assurément un élément déterminant de cette capacité, bien sûr à travers l'émergence du lexique (les objets et concepts) mais aussi par celle de la syntaxe (les règles de formation des phrases) et de la grammaire (qui définit et structure les relations entre objets). Les travaux de Chomsky, dès la décennie 1960, concluent à l'existence d'une grammaire universelle « câblée » dans le cerveau humain afin, notamment, d'expliquer la capacité d'un nourrisson à acquérir le langage de la culture qui l'entoure (quelle qu'elle soit). Ainsi, nous avons tous la même manière fondamentale d'encoder et de transmettre l'information, les différences de langues étant finalement superficielles.

Chomsky a contribué de façon majeure à la linguistique en l'inscrivant dans le cadre théorique des **langages formels**. Un langage formel est un langage soumis à des règles sémantiques, syntaxiques et grammaticales tellement strictes qu'il peut être entièrement décrit et étudié par les mathématiques.



Noam Chomsky (1928–)
Linguiste et philosophe étasunien, qui a classé les langages formels en fonction de leur pouvoir d'expression des langages humains.

Il s'agit donc d'un *modèle* de langue, une nécessaire simplification qui permet d'en faire une étude scientfiique.

Exemple 1.1

Un exemple de langage formel est celui des **formules mathématiques**. En utilisant le lexique des opérations arithmétiques $+$, $-$, $\times$, $\div$ et deux symboles X et Y, nous pouvons fabriquer des expressions en juxtaposant des caractères :

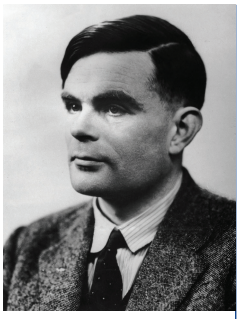
$$X - X + Y \times X, \qquad X + X \div Y - X \times Y - X, \qquad XY + -X \div \times \times YY \times \times \times.$$

Avant même de vouloir manipuler ces expressions, il convient de savoir si elles sont bien construites, pour que le lecteur puisse tenter d'en tirer du sens. Il faut donc se mettre d'accord sur une *syntaxe*. Une expression sera valide dans le langage $\mathcal{L}$ (ce sera une *formule du langage* $\mathcal{L}$) si les règles suivantes sont respectées :

1. deux symboles sont toujours séparés par une opération,

2. une opération est toujours encadrée par deux symboles.

Ces règles disqualifient la troisième expression ci-avant, mais valident les deux premières.



Alan Turing (1912–1954) Mathématicien et cryptologue anglais, fondateur de l'informatique moderne.

Exemple 1.2

Une propriété du langage $\mathcal{L}$ précédent est qu'une formule non vide commence et se finit toujours par un symbole. C'est un *invariant* du langage. Bien qu'il ne soit pas très intéressant de le savoir, cela fournit un moyen de comparer ce langage formel aux autres. Considérons le langage $\mathcal{L}'$ obtenu en rajoutant aux caractères précédents les parenthèses (,) et en prenant les règles connues du bon parenthésage. Par exemple, l'expression

$$((X + Y) \times Y) - Y \div (X + Y)$$

est une formule du langage $\mathcal{L}'$. Elle ne commence pas par un symbole, donc $\mathcal{L}'$ ne respecte pas l'invariant de $\mathcal{L}$: les deux langages sont différents. Ils n'ont d'ailleurs pas le même pouvoir d'expression : autant $(X + Y) \times Y$ peut s'exprimer comme $X \times Y + Y \times Y$ dans $\mathcal{L}$, autant $Y \div (X + Y)$ n'a pas d'équivalent dans $\mathcal{L}$.

Exemple 1.3

Un **langage informatique**, comme le langage Python, est un exemple de langage formel. Il a un lexique bien défini (les **mots-clefs** du langage : `def`, `if`, `for`, `else`, `import`, etc.) et une syntaxe propre répondant à des spécifications précises. La connaissance de ces règles permet aux programmeurs de produire un **code source**, c'est-à-dire une succession de caractères (un texte) qui forment une suite valide d'instructions pour le langage Python. Cela signifie qu'un interpréteur Python peut lire le code source et en *exécuter* les instructions (actions concrètes réalisées par l'ordinateur).

FOCUS

Les machines de Turing et les langages complets

Certains langages informatiques sont plus adaptés pour décrire certaines actions que d'autres langages. Par exemple, l'assembleur est un langage très bas niveau (très proche du processeur et de l'architecture matérielle), alors que le langage Python est un langage très haut niveau (programmation objet abstraite, totalement indépendante de l'architecture matérielle).

Il est parfois possible de convertir (traduire) un code écrit dans un langage en un code d'un autre langage réalisant les mêmes tâches. Bien que chaque langage informatique ait sa spécificité, nous nous attendons néanmoins à ce qu'un langage soit capable d'exprimer certaines opérations : calcul arithmétique, lecture / écriture mémoire, boucle et branchement conditionnel, appel de fonctions. . . Ces fonctionnalités de base sont représentées de manière abstraite par le concept de **machine de Turing**. Nous dirons qu'un langage de programmation est **complet** (au sens de Turing) s'il permet de simuler une machine de Turing. C'est le cas de la plupart des langages haut niveau (C, C++, Java, Lolcode, Python, etc.) et, bien sûr, de l'assembleur.

Une machine de Turing est donc un outil théorique qui permet de comparer les capacités de différents langages formels. C'est un *modèle basique d'ordinateur universel*, qui rassemble le minimum des fonctionnalités qu'un « calculateur autonome » doit posséder pour mériter ce titre.

1.2 Des nombres qui comptent

Même si tout cela se déroule dans les organes dédiés du corps (sans que nous sachions vraiment comment), les fonctions cognitives n'en sont pas moins des processus abstraits agissant sur des objets abstraits. Bien sûr il paraît étrange de qualifier d'abstraits des faits aussi concrets que « il y a trois souris », mais la dernière fois que j'en ai parlé à mon chat – et bien qu'il sache probablement compter

© Dunod. Toute reproduction non autorisée est un délit.

jusqu'à trois – il n'a pas compris où je voulais en venir (il a continué à ronronner poliment). Le langage humain permet d'isoler et de véhiculer un concept nouveau : celui de « trois ». Trois arbres, trois souris, trois cailloux. . . tous ces paquets sont équivalents, ils « comptent » la même quantité.

Remarque. Comme nous allons progressivement monter en gamme dans l'abstraction, entendons-nous malgré tout pour qualifier de *concrets* les **entiers naturels**, c'est-à-dire les nombres qui comptent des quantités (indépendamment de leur nature). Nous les prendrons comme objets basiques dans tout ce qui suit, et supposerons que le lecteur est familier avec l'écriture des nombres entiers avec des chiffres (numération positionnelle, rappelée en Définition 1.12).

Exemple 1.4

L'ensemble $\{a, b, c, d\}$ comporte autant d'éléments que l'ensemble $\{\heartsuit, \spadesuit, \diamond, \clubsuit\}$. Ce cardinal peut être *représenté* (écrit) par :

– `4` en écriture décimale,

– `100` en écriture binaire,

– `IV` en écriture romaine,

– `• • • •` en écriture maya.

Remarque. Certaines cultures écrivent différemment les quantités en fonction de la nature des objets comptés ou du contexte : nous pourrons rencontrer dans un même texte l'évocation de 19 maisons qui ont brûlé un même jour du XIX^{e} siècle à dix-neuf heures.

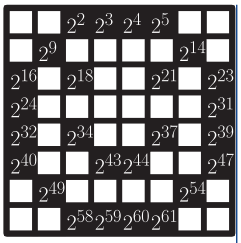
FOCUS

Nombre et écriture

Le nombre représenté est apparu avec l'écriture, dans les premiers grands États organisés, à différentes époques et en différents lieux entre −4 000 et −1 000. Son utilité première est de rendre possible l'administration des biens et ressources (décompte des têtes de bétail, des récoltes. . .) et la gestion du temps (calendriers des saisons, des rituels, astronomie. . .) à l'échelle d'une population grandissante (la cité-État d'Uruk était peuplée de plus de 100 000 habitants vers −2500). L'écriture d'un nombre permet des échanges d'informations plus simples, sûrs et rapides entre différentes parties administrées, ce qui constitue un enjeu majeur à l'époque de la tablette d'argile mais aussi à celle de la fibre optique.

1.3 Stockage et encodage sont les deux mamelles de l'informatique

Nous pouvons décider qu'un entier naturel représente autre chose qu'une quantité cardinale. Par exemple, le nombre 4 342 201 926 862 324 284 peut représenter l'image monochrome d'un smiley (voir ci-contre). Ou, peut-être, un fragment de la chanson « Libérée, délivrée. . . ». Ou pire.



■ $2^2+2^3+2^4+2^5+2^9+\cdots=4\ 342\ 201\ 926\ 862\ 324\ 284$ (codable en 64 bits)

Définition 1.5

L'**encodage** d'une information est sa représentation par un nombre (les **données**) et par un algorithme de traduction (le **format**), ce dernier étant indépendant des données et permettant de les transformer en les informations initiales et *vice versa*.

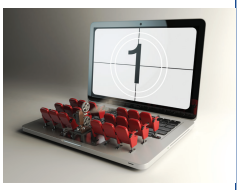
Du point de vue du mathématicien, la totalité de l'information encodée dans le disque dur d'un ordinateur (ou dans sa mémoire vive) est un nombre N entier – probablement gigantesque – puisqu'elle est finalement écrite comme une très longue succession de chiffres binaires. L'organisation de cet espace de stockage en table des matières, différents chapitres (appelées *partitions*) pour organiser les décimales du nombre N (les données) est lui-même soumis au format dicté par le système d'exploitation (`fat` , `ext`, `ntfs`, `hpfs` . . .). Le mathématicien n'a pas besoin de se soucier de ces choix, il fait entièrement confiance à l'informaticien.

Exemple 1.6

Une carte mémoire 64 GB peut être imaginée comme une succession de $64\times 2^{30}\times 8=2^{39}=549\ 755\ 813\ 888$ cases pouvant chacune prendre deux états : pleine (1) ou vide (0). Alors le contenu de la carte SD correspond exactement à un nombre N compris entre 0 (toutes les cases contiennent 0) et $2^{549\ 755\ 813\ 888}-1$ (toutes les cases contiennent 1).

1.4 Calcul, calculateurs et ordinateurs

L'encodage d'informations diverses sous forme de nombres permet de ramener à du calcul arithmétique bon nombre de tâches, notamment liées à l'activité professionnelle, au divertissement et à la communication.



Exemple 1.7

Le streaming vidéo permet l'envoi d'images à travers les réseaux de télécommunication. Entre l'acquisition de l'image par la caméra et sa restitution par l'écran, plusieurs étapes sont nécessaires, comme décrit par la figure ci-après.

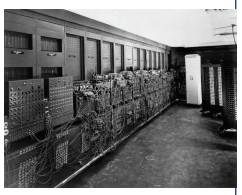
■ Le mot « calcul » partage la racine latine *calculi* avec « cailloux ».

L'étape de numérisation des images, puis sa reconstruction par le lecteur vidéo, utilise des mathématiques assez sophistiquées (analyse puis synthèse spectrale par transformée de Fourier), qui sont aussi utiles pour la compression. Il est nécessaire de compresser l'information après l'avoir encodée : une vidéo non compressée 1080p en 24 images par seconde consomme 145 Mo/s de débit, ce qui excède la capacité des réseaux actuels. Après compression, ce débit n'est que de 0,5 Mo/s, pour une perte de qualité assez réduite. C'est magique ? Non, c'est mathématique.

© Dunod. Toute reproduction non autorisée est un délit.

Machine à additionner à engrenages, avec gestion de la retenue, appelée « pascaline » et crée par PASCAL en 1650.

John VON NEUMANN (1903–1957) Mathématicien et physicien étasunien d'origine hongroise, père de l'architecture moderne des ordinateurs.



ENIAC, premier calculateur programmable, entièrement électronique, construit en 1945 par ECKERT et MAUCHLY. Il contient 17 468 tubes à vide, pèse 30 tonnes et occupe 167 m^2. Il calcule en base 10 et effectue 100 000 additions par seconde grâce à 20 modules de calcul parallèles.

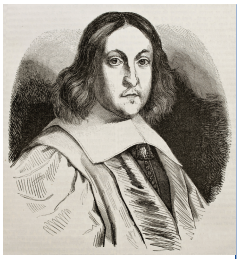
La simplicité apparente d'une tâche ne présage pas de la complexité des calculs qui seront à faire, et il devient vite nécessaire d'utiliser des outils aidant à calculer. L'utilisation d'un outil permet aussi d'évacuer les erreurs de calcul introduites par les humains (à condition bien sûr que le matériel et la recette ne contiennent pas de « bugs ») et de gagner en rapidité. Les premiers outils ont consisté en un support permettant de poser les opérations (abaques) ou de stocker les résultats intermédiaires, comme les bouliers utilisés depuis l'Antiquité ou, plus ancien et plus rudimentaire, des entailles dans un bâton, des tas de cailloux. . .

Les mécanismes autonomes (notamment les engrenages des moulins) ont inspiré les premières machines à calculer mécaniques : de la pascaline du milieu du XVII^e siècle aux serveurs de calculs massivement parallélisés du début XXI^e siècle, l'humain a automatisé le calcul dès qu'il l'a pu. Il ne faut toutefois pas confondre une **machine à calculer**, qui peut seulement réaliser des opérations arithmétiques, avec une **machine programmable**, c'est-à-dire une machine effectuant des opérations (notamment arithmétiques, mais pas seulement) dans l'ordre choisi par l'utilisateur et spécifiées à la machine par un codage particulier.

Les premières machines programmables ne servaient d'ailleurs pas à calculer. Elles sont apparues avec les métiers à tisser de JACQUARD au tout début du XIX^e siècle et se programmaient grâce à des cartes perforées. Cela a beaucoup inspiré l'Anglais BABBAGE, dont la « machine analytique » – imaginée à partir des années 1830 – est considérée comme le premier ordinateur. C'est pour cette machine que LOVELACE a écrit le premier programme informatique de l'humanité. Le principal défaut de cette machine est qu'elle n'a pu être réalisée qu'au début du XX^e siècle, par un des fils BABBAGE.

Les tous premiers calculateurs programmables fonctionnels, appelons-les des **ordinateurs** pour simplifier le propos, sont assez vite devenus électromécaniques, puis entièrement électroniques (en particulier pour des questions de fiabilité matérielle et de rapidité des calculs). Nous pouvons considérer que ceux-ci ont été développés grâce aux machines utilisées pendant la Seconde Guerre mondiale pour s'occuper de *cryptographie*, c'est-à-dire pour crypter et décrypter des textes écrits en langage naturel. Les versions réellement efficaces utilisées dès la décennie 1950 sont basées sur l'architecture de VON NEUMANN (encore utilisée auourd'hui). Avec d'autres jeunes mathématiciens anglais, TURING a travaillé secrètement pour les services britanniques afin de décoder le chiffrage allemand produit par la machine Enigma, en contribuant de manière majeure au cassage des codes nazis. Le langage a donc constitué le premier champ d'application des premiers ordinateurs, et de nos jours la cryptographie des données numériques est une brique essentielle des protocoles de communication et de stockage.

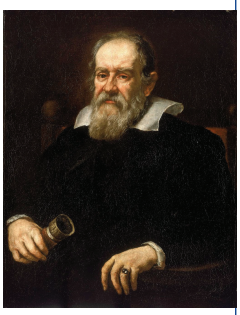
2 Calculer avec des nombres et des symboles



Pierre de Fermat (≃1605–1665)
Magistrat et mathématicien français.

Simultanément à la faculté de communiquer des informations immatérielles, notre espèce a développé les capacités d'analyser cette information, prévoir et agir en conséquence. C'est le fait culturel dévolu à la science et à la technologie, dont les mathématiques et l'informatique sont le bras armé.

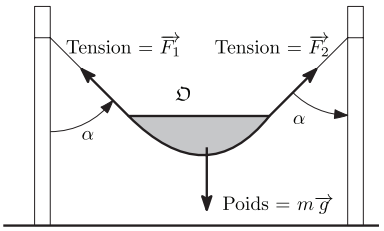
Certaines découvertes mathématiques furent fondatrices. Ainsi l'avènement du calcul des dérivées, porté par Fermat, Leibniz et Newton, a accouché de la création des sciences physiques (un travail déjà entamé par Galilée) en les sortant des dépendances de la philosophie naturelle pour les consacrer comme disciplines à part entière. Dès lors, les scientifiques ont recherché des *modèles* mathématiques dont les *prévisions* permettaient de rendre compte de l'*expérience*. Ces modèles sont en général des équations décrivant les contraintes existant entre différentes quantités mises en jeu par le système étudié.



Galilée (1564–1642)
Savant italien.

Exemple 1.8

La mécanique modélise une « force » exercée sur un objet $\mathfrak{O}$ par un vecteur $\vec{F}$. Bien que je n'ai jamais vu de petite flèche faire tomber une pomme, la théorie mathématique des vecteurs décrit parfaitement les observations que nous faisons au quotidien. En particulier, si je sais additionner deux vecteurs $\vec{F_1} + \vec{F_2}$, alors je suis capable de *prédire* l'effet qu'aura l'application simultanée à un objet des deux forces correspondant à $\vec{F_1}$ et $\vec{F_2}$.



La loi fondamentale de la statique dans un référentiel galiléen s'énonce sous la forme suivante :

$$\text{un objet } \mathfrak{O} \text{ est au repos} \Leftrightarrow \sum_{\vec{F} \text{ appliquée à } \mathfrak{O}} \vec{F} = \vec{0}.$$

Dans le cas d'une masse suspendue à deux cordes, comme un hamac attaché par des cordes de longueur 1 m (voir figure ci-avant), cette équation impose des relations sur l'angle α formé avec la verticale et la norme T des forces de tension. Si un prof de maths de masse 92 kg est au repos dans le hamac, la loi de la statique impose :

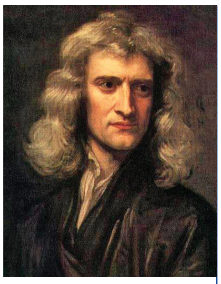
$$T = \frac{450}{\cos(\alpha)}.$$

La valeur attendue de T permet de dimensionner au préalable les cordes utilisées pour soutenir le corps au repos (plus la tension est grande, plus la corde doit être résistante).

Gottfried Leibniz (1646–1716)
Savant et philosophe allemand.

Plus généralement, la mise en œuvre de mes facultés calculatoires abstraites me donne des informations sur le devenir concret d'un système : résoudre les équations régissant un système permet donc d'en prévoir l'évolution. La qualité de la prédiction dépend de ma capacité à organiser et mener des calculs complexes, nombreux, mais aussi à contrôler l'erreur commise lors des approximations. Ce calcul peut être mené à différents niveaux.

© Dunod. Toute reproduction non autorisée est un délit.



Isaac NEWTON (1643–1727)
Savant et philosophe britannique.

Calcul formel : nous cherchons une *solution exacte*, c'est-à-dire une formule mathématique (au sens du chapitre 3) qui résout l'équation de départ ; cette tâche est parfois difficile, voire impossible.

Calcul numérique : à défaut (ou pour d'autres raisons, dont la rapidité de calcul), nous calculons des valeurs numériques approchant la solution exacte.

Exemple 1.9

Reprenons l'exemple du hamac. La connaissance de la formule $T = \frac{450}{\cos(\alpha)}$ nous enseigne comment évolue la tension des cordes en fonction de l'angle : si on souhaite avoir un hamac complètement horizontal ($\alpha = 90°$), alors il faut appliquer une tension infinie. En général, l'installation (l'emplacement des arbres et la longueur des éléments) impose l'angle α, disons $\alpha = 63°$, mais cet angle n'est pas une « valeur facile » et la valeur exacte de $\cos(63°)$ n'est pas connue. Il nous faut donc trouver un moyen de déterminer une valeur approchée de $\cos(63°)$ grâce au calcul numérique.

L'encodage de l'information par un ordinateur permet de comprendre des nombres comme autre chose que des nombres. Cette faculté a ouvert très tôt la possibilité d'aller au-delà du calcul numérique, comme l'avait entrevu la pionnière LOVELACE plus d'un siècle avant la réalisation concrète des premiers ordinateurs.

> « *De nombreuses personnes, qui connaissent mal les études mathématiques, pensent que comme le travail de la Machine est de donner des résultats en notation numérique, la nature du processus doit forcément être arithmétique et numérique, plutôt qu'algébrique et analytique. C'est une erreur… La Machine peut produire trois types de résultats : […] symboliques […] ; numériques […] ; et algébriques en notation littérale.* »
>
> LOVELACE, parlant de la machine analytique de BABBAGE (voir section 1.4).

Un ordinateur peut donc nous aider à faire du calcul numérique mais aussi du calcul symbolique (dans lequel nous incluons le calcul « algébrique en notation littérale »).

2.1 De l'algèbre au calcul symbolique

Même si le lecteur sait qu'une dérivée se *définit* comme la limite d'un taux d'accroissement, il sait également que ce n'est pas ainsi qu'une dérivée se *calcule*. Il existe des règles permettant de calculer la dérivée d'une formule à partir de la dérivée, supposée connue, des éléments constituant la formule ainsi que celles des opérations arithmétiques et de composition.

Exemple 1.10

Pour calculer la dérivée de la fonction usuelle $f : x \mapsto \cos(x)^4$, je considère la formule $\cos(X)^4$ et j'applique la règle de dérivation d'une puissance :

$$\frac{\mathrm{d}\cos(X)^4}{\mathrm{d}X} = 4 \times \left(\frac{\mathrm{d}\cos(X)}{\mathrm{d}X}\right) \times \cos(X)^3 = -4\sin(X)\cos(X)^3.$$

Alors je peux affirmer que pour chaque x réel on a $f'(x) = -4\sin(x)\cos(x)^3$. À aucun moment je n'ai calculé la limite d'un taux d'accroissement. Toutefois, la règle $\cos' = -\sin$ est une conséquence d'un tel calcul, mais il suffit de l'avoir fait une fois pour toutes et de garder le résultat en mémoire (chapitre 7) : c'est un *théorème mathématique*.

Jean Sammet (1928–2017) Informaticienne étasunienne, créatrice d'un des premiers langages de calcul formel.

Un programme permettant de manipuler des mots et des formules d'un langage formel s'inscrit dans le cadre du **calcul formel et symbolique**. Le premier langage informatique permettant de manipuler des formules mathématiques a été créé par l'informaticienne Sammet dans les années 1960. Ce type de calcul s'inscrit dans le champ mathématique appelé **algèbre**.

Exemple 1.11

Le langage Python dispose d'une bibliothèque, `sympy`, dédiée au calcul symbolique. L'exécution du code suivant :

```
1 from sympy import *
2
3 X= Symbol ("x")
4 F= exp(-X*X)
5 print (f "La dérivée de {F} est {diff(F,X)}")
```

produit la sortie : `La dérivée de exp(-x**2) est -2*x*exp(-x**2)`. Nous montrerons au chapitre 7 comment coder soi-même une bibliothèque rudimentaire pour effectuer des calculs automatiques de dérivées; en dehors de cette démonstration de principe (qui sera un peu décevante), nous nous appuierons sur la bibliothèque `sympy`.

2.2 De l'analyse au calcul numérique

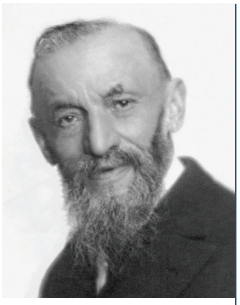
Comme nous en discutons dans le focus p. 175, il n'est pas toujours possible de travailler avec des formules pour résoudre un problème mathématique de manière exacte. Par exemple, aucune primitive $\int \exp(-X^2)\,dX$ de la gaussienne ne peut s'exprimer comme une formule (au sens du chapitre 3).

Nous sommes donc contraints, pour bon nombre de calculs issus de situations concrètes, de recourir à des techniques de calcul numérique approché. Le domaine mathématique couvrant ces questions est celui de l'**analyse** : c'est d'abord la construction et l'étude théorique des *nombres réels* et des quantités qui dépendent de nombres réels (les *fonctions*), mais aussi, de manière plus pratique, la conception des codes permettant de calculer des *valeurs approchées* de nombres, de fonctions, de solutions d'équations…

Un des enjeux principaux de l'analyse numérique est celui de *contrôler l'erreur d'approximation*, c'est-à-dire être capable d'affirmer mathématiquement que les décimales qui ont été calculées par l'ordinateur sont *exactes*. L'outil que nous emploierons pour valider ce contrôle est le **certificat de convergence** (définition 4.8).

© Dunod. Toute reproduction non autorisée est un délit.

3 Représenter et calculer avec des nombres



Giuseppe PEANO (1858–1932)
Mathématicien et linguiste italien.

L'objet initial de notre théorie est le nombre entier naturel. Nous l'avons introduit comme une manière de compter le nombre d'éléments d'un ensemble, et nous utilisons son écriture dans un système de *numération positionnelle* dans une base donnée (en général 10, mais nous trouverons bien sûr les bases 2 et 16 sur notre route). Nous explorons ici comment les nombres entiers, et les autres, peuvent être stockés et manipulés par une machine.

Nous avons proposé ci-avant un cadre dans lequel traiter des questions de « calculabilité », celui des machines de Turing. Une machine de Turing est un système abstrait qui permet de réaliser des tâches basiques permettant de construire, par exemple, les opérations arithmétiques sur les entiers. Pour faire simple, et afin d'utiliser un vocabulaire plus familier au lecteur, nous parlerons d'*ordinateur* et de *programme informatique*. Néanmoins, la théorie que nous souhaitons développer nécessite de recourir à un ordinateur idéal qui peut stocker beaucoup d'informations.

3.1 Les nombres entiers

FOCUS

Axiomatique de PEANO

Une manière moderne de construire les entiers naturels – proposée par PEANO à la fin du XIX^e siècle – est de considérer une origine abstraite, que nous notons 0, et une « fonction successeur » que nous notons τ. Cette fonction successeur permet de construire le « nombre » suivant, noté $\tau(0)$, puis celui qui suit encore sera $\tau(\tau(0))$, etc. En plus de *compter* les nombres, cette notion de successeur permet de les *ordonner*, en définissant le classement

$$0 < \tau(0) < \tau(\tau(0)) < \tau(\tau(\tau(0))) < \cdots$$

Nous choisissons bien sûr de noter $1 := \tau(0)$ le successeur de 0, et plus généralement le successeur de l'entier n est noté $n+1$ plutôt que $\tau(n)$ (de même, l'antécédent d'un entier $n > 0$ est noté $n-1$). Cette notion un peu abstraite n'est pas complètement dénuée de lien avec la réalité : c'est bien ainsi que nous compterions le nombre de spaghettis dans une boîte – un soir de grand ennui– les uns après les autres. Même si nous pouvons dénombrer d'un seul regard les petites quantités (le nombre de points sur une face de dé, par exemple), nous revenons au comptage un par un dès que les éléments sont nombreux.

L'histoire des nombres est très étroitement liée à leur **représentation** (système de numération). Chaque grande culture a disposé d'un système de numération, qui ne passe pas nécessairement par l'écrit. Parmi ces cultures, citons les civilisations grecque, romaine, chinoise, maya, aztèque, sumérienne, babylonienne, etc.

Dans l'Antiquité grecque puis romaine, les nombres n'étaient pas écrits avec des **chiffres** dans une système de **numération positionnelle**, comme aujourd'hui. La représentation moderne, apparue en Europe vers la fin du Moyen Âge, est le croisement du système positionnel babylonien et des chiffres indiens. Il se nomme système de numération **indo-arabe**, du fait de son introduction en Europe suite aux échanges entre savants européens et savants arabes, ces derniers ayant bénéficié de

l'influence des mathématiques indiennes. C'est le système de numération le plus répandu à l'heure actuelle, et c'est celui que nous utiliserons.

FOCUS

Le système de numération par position

Dans un tel système de numération, la **position** d'un chiffre dans l'écriture du nombre lui donne un poids plus ou moins grand (le chiffre des unités, des dizaines, des centaines, etc.). Ces systèmes sont apparus car ils permettent de « poser » les opérations arithmétiques, c'est-à-dire qu'ils offrent un algorithme de calcul. L'exemple suivant montre que le choix du système de représentation influe sur la facilité à effectuer les opérations.

XIX	19
+MCIV	+1104
=??????	= 1123

Définition 1.12

Soit $b > 0$ un entier.

1. La **représentation** (ou écriture) en **base** b d'un nombre entier, est une chaîne de caractères, chacun des caractères étant appelé **chiffre**. Le cas échéant, pour indiquer le signe de ce nombre, la chaîne de caractère peut être préfixée par le caractère `+` ou `-`.
2. Les **chiffres du système binaire** (base 2) sont les caractères `"0","1"`.
3. Les **chiffres du système décimal** (base 10) sont les caractères `"0","1","2","3","4","5","6","7","8","9"`.
4. Les **chiffres du système hexadécimal** (base 16) sont les caractères `"0"` `,...,` `"9","A","B","C","D","E","F"`.
5. Pour distinguer la représentation d'un même nombre dans différentes bases, nous utiliserons la **notation** $(\text{signe chiffres})_{\text{base}}$.
6. Pour chaque chiffre c, nous choisissons une valeur entière. Pour faire simple, la valeur des chiffres habituels correspond aux valeurs standard, celle du nombre (à un chiffre) correspondant :

$$\texttt{"0"} \mapsto 0 \qquad \texttt{"1"} \mapsto \tau(0) = 1 \qquad \texttt{"2"} \mapsto \tau(\tau(0)) = 2\ldots$$

Pour l'hexadécimal, la valeur 10 est donnée à `"A"`, la valeur 11 à `"B"`, etc. jusqu'à 15 pour `"F"`. Pour alléger les notations, nous noterons cette valeur également par c.

7. La **valeur** de l'écriture $(\pm c_m \cdots c_0)_b$ se calcule alors comme

$$(\pm 1) \times (c_m b^m + c_{m-1} b^{m-1} + \cdots + c_1 b + c_0) = \pm \sum_{j=0}^{m} c_j b^j .$$

Exemple 1.13 Représentation dans différentes bases

Voici différentes représentations du nombre de lettres de l'alphabet français :

$$(26)_{10} = 2 \times 10 + 6 = 26$$
$$(11\,010)_2 = 2^4 + 2^3 + 2^1 = 26$$
$$(1A)_{16} = 1 \times 16 + 10 = 26.$$

© Dunod. Toute reproduction non autorisée est un délit.

Remarque. L'architecture des ordinateurs actuels privilégie le système binaire comme base de représentation et de traitement des nombres par les composants électroniques. Toutefois, la représentation courante (dans les codes et les programmes) des nombres destinés à l'ordinateur (nombre de couleurs, tailles d'images, etc.) est hexadécimale. Pour ne pas la confondre avec l'écriture décimale, elle est préfixée, dans certains codes, par `0x`, comme dans `0x1F2C` = 7980.

3.2 Des barrettes de RAM plein les poches

Un ordinateur stocke un nombre entier dans sa mémoire en écrivant la succession de ses chiffres binaires. Dès lors, la taille de la mémoire limite la taille du plus grand nombre qu'elle peut stocker, comme discuté dans l'exemple 1.6. Si nous souhaitons écrire un nombre très grand, il faut une très grande mémoire. Quel nombre pourrait-on écrire si nous mettions bout à bout l'ensemble des supports de stockage de l'humanité? Chaque disque dur, chaque giga-octet de RAM... Et si nous consacrions chaque atome de notre planète à stocker un chiffre binaire, quel nombre obtiendrions-nous? Et avec notre galaxie? Il est difficile de répondre à ces questions, et il serait également difficile de réaliser ce que représente concrètement la réponse à ces questions. À quel point $100^{(1\,000^{10\,000})}$ est un grand nombre? Y a-t-il plus ou moins de $100^{(1\,000^{10\,000})}$ atomes dans notre corps? Dans notre galaxie?

Quoi qu'il en soit, nous voyons que les moyens actuels de stockage ressemblent beaucoup à une mémoire « infinie », c'est pourquoi il paraît crédible, pour mener notre étude théorique, de considérer que nous disposons d'un ordinateur avec une mémoire pouvant s'agrandir en fonction des besoins.

Attention. Dans toute la suite de l'ouvrage, nous travaillons donc avec un ordinateur ayant une **mémoire potentiellement infinie**. Ainsi, pour tout nombre entier que nous serons amenés à considérer, il sera possible de mobiliser assez de capacités pour écrire le nombre dans la mémoire de l'ordinateur.

La règle du jeu interdit d'écrire en une seule fois une quantité infinie d'information.

3.3 Calculer, c'est quoi au juste ?

Turing a non seulement développé un cadre théorique qui permet de poser cette question et d'y répondre mathématiquement, mais il a également posé des questions de nature plus philosophique sur la calculabilité et sur la nature des processus cognitifs humains.

FOCUS

La thèse de Church-Turing

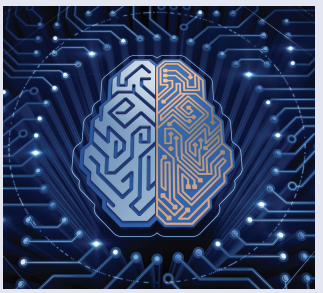
Cette « thèse », c'est-à-dire une posture philosophique à prendre plutôt comme une question, stipule que :

« *Tout ce qui est calculable* [dans un sens non défini précisément] *peut être effectué par un ordinateur.* »

Il existe des versions plus ou moins fortes de cette thèse, en fonction de ce qui est entendu par « calculable ». Au sens le plus faible de « calculé = produit par un ordinateur », nous obtenons une tautologie ; c'est la définition du mathématicien (toujours peu enclin à prendre des positions philosophiques trop radicales). Mais nous pourrions tout autant affirmer que le monde qui nous entoure est dans son intégralité le fruit d'un calcul, comme le sous-tend notre tendance exagérée à tout modéliser mathématiquement. La thèse affirme dans ce cas-là que la « réalité » pourrait être intégralement simulée (ce thème étant une source inépuisable d'inspiration pour la science-fiction). En poussant le curseur encore plus loin, le fonctionnement même de notre corps, ses pensées et son sentiment de « conscience individuelle », voire de « transcendance » d'une partie de son être, entre dans le champ de la question posée par CHURCH et son élève TURING :

Un ordinateur peut-il simuler « l'âme » humaine ?

Cette problématique accompagne l'émergence de l'intelligence artificielle, certes, mais va bien au-delà en nourrissant directement les débats très actuels autour du « transhumanisme ». Dans ce cadre, il n'est pas étonnant d'observer que les tenants les plus médiatisés de cette doctrine sont ceux et celles qui dirigent les plus grandes firmes de l'industrie informatique, dont le principal moteur économique est le commerce automatisé des publicités profilées et des données personnelles (celles des autres, de préférence).



Pour faire simple, nous considérons qu'une *machine de Turing* est un programme informatique s'exécutant sur un ordinateur disposant d'une mémoire potentiellement infinie.

Définition 1.14

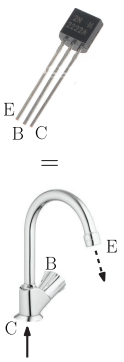
Une fonction f prenant des nombres entiers $n_1, \ldots, n_k$ en entrée et produisant le nombre entier $f(n_1, \ldots, n_k)$ en sortie est dite **calculable** lorsqu'il existe un programme informatique M qui vérifie la propriété suivante : pour tout choix d'entiers $n_1, \ldots, n_k$ donnés comme valeurs d'entrée de M, l'exécution de M se termine, à la suite de quoi la mémoire contient le nombre $f(n_1, \ldots, n_k)$.

En pratique. Une fonction calculable est une fonction dont les valeurs se calculent avec un ordinateur.

Exemple 1.15

L'addition de deux entiers naturels est une fonction calculable, en partant du principe que sont connus le successeur $n + 1$ et l'antécédent $n - 1$ de n'importe quel entier n.

```
1  def add (n1, n2):
2      if n2==0:
3          return n1
4      else:
5          return add (n1, n2-1)+1
```



Un transistor se comporte comme un robinet : l'eau arrive au collecteur C, et si la base B est alimentée alors le courant ressort par l'émetteur E.

© Dunod. Toute reproduction non autorisée est un délit.

De même, la multiplication d'entiers naturels est une fonction calculable, et plus généralement les autres opérations arithmétiques le sont.

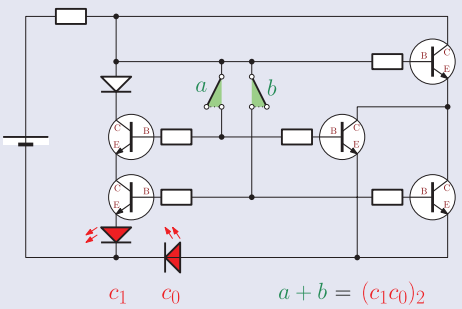
```
1 def mult (n1, n2):
2     if n2==0:
3         return 0
4     else:
5         return add (n1, mult (n1, n2-1))
```

FOCUS

Les additions électroniques

Disons quelques mots sur les réalisations matérielles possibles d'un processeur. Les processeurs modernes sont constitués de centaines de milliards de transistors, habilement connectés et savamment entrecoupés de résistances. Le circuit électronique ci-dessous est un *additionneur*. Les différentes positions prises par les interrupteurs a et b activent le résultat $(c_1c_0)_2$ en binaire, selon la table ci-après.

$a+b$	0	1
0	$(00)_2$	$(01)_2$
1	$(01)_2$	$(10)_2$



(Nous laissons le lecteur généraliser ce circuit pour additionner deux nombres codés sur 64 bits.) Bien que nous ayons l'impression que ce circuit « fait quelque chose », il n'est finalement que la transcription sous forme de silicone de la table d'addition donnée à gauche. L'unité de calcul arithmétique logée dans un processeur n'est en quelque sorte qu'un (énorme) recueil de tables d'addition, de mulitplication, de logarithmes, etc.

Définition 1.16

Puisqu'un ordinateur peut :

— stocker n'importe quel nombre entier,

— appliquer les opérations arithmétiques entre nombres entiers arbitraires,

— décider si deux entiers sont égaux,

nous dirons que les entiers sont **complètement calculables**.

Nous présentons dans les sections suivantes deux classes de nombres très importantes, qui sont également complètement calculables : les rationnels (et donc les décimaux), mais aussi les « nombres algébriques ». Nous discutons ensuite brièvement en quoi ces nombres ne sont pas suffisants pour alimenter notre science et notre technologie, ce qui nous conduira à introduire les nombres réels qui ne sont, eux, pas complètement calculables.

3.4 Calculer avec des décimaux

Quand un ordinateur fait des calculs avec des nombres à « virgule flottante », il utilise en fait des décimaux représentés par des entiers (selon la notation scientifique binaire) :

$$(-1)^{\text{signe}} \times \text{mantisse} \times 2^{\text{exposant}}.$$

La norme IEEE 754 décrit les spécifications pour les manipulations de nombres flottants dans les processeurs actuels. En particulier, voici comment sont encodés les flottants dans les architectures courantes.

Précision	Encodage	Signe	Exposant	Mantisse	Valeur	Chiffres significatifs
Simple	32 bits	1 bit	8 bits	23 bits	$(-1)^s \times m \times 2^{e-127}$	$\simeq 7$
Double	64 bits	1 bit	11 bits	52 bits	$(-1)^s \times m \times 2^{e-1023}$	$\simeq 16$

Exemple 1.17

Le nombre binaire `0 00111010 01100011011001101100011` représente dans l'encodage IEEE 754 simple précision le nombre donné par

$$\begin{aligned} \text{signe} &= 0 \\ \text{exposant} &= \texttt{00111010} = 58 \\ \text{mantisse} &= \texttt{01100011011001101100011} = 3\,257\,187, \end{aligned}$$

c'est-à-dire

$$(-1)^0 \times \frac{3\,257\,187}{2^{69}} \simeq 0{,}000\,000\,000\,000\,005\,517\,889\,408\,736\,786.$$

Remarque. Le langage Python implémente l'utilisation de nombres entiers de taille arbitraire, qui n'est pas limitée par l'architecture du processeur (32, 64 ou 128 bits) mais seulement par la mémoire :

```
1 print (2**200)
2 # 1606938044258990275541962092341162602522202993782792835301376
```

En autorisant des mantisses et des exposants très grands, nous obtenons des flottants avec beaucoup de décimales. Nous pouvons donc stocker et faire des calculs avec tous les décimaux grâce à un ordinateur.

En pratique. Les nombres flottants sont affichés en notation scientifique décimale dans Python. Ainsi le nombre $0{,}000\,000\,023\,441\,2 = 2{,}34412 \times 10^{-8}$ sera affiché avec un `e` à la place du $\times 10$.

```
1 print (0.0000000234412)          #2.34412e-08
```

© Dunod. Toute reproduction non autorisée est un délit.