Michaël Baudin

Méthodes numériques avec Python

Théorie, algorithmes, implémentation et applications avec Python 3

DUNOD

Graphisme de couverture : Elizabeth Riba
Illustration de couverture : © Lauren Suryanata - Shutterstock.com

11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-084079-3

Avant-propos

Le calcul scientifique en Python

L'objectif de ce manuel est de présenter les principes mathématiques, les applications et la mise en œuvre de méthodes de calcul scientifique en Python. Ce texte évoque, tour à tour, la pratique et la théorie : nous montrons l'utilisation de ces méthodes dans le langage Python en utilisant les librairies `NumPy` et `SciPy` et l'analyse théorique sur laquelle le calcul s'appuie.

Partout où cela est possible et pertinent, nous présenterons des applications réelles plutôt que des exemples simplifiés ou théoriques. En effet, l'objectif de ce texte est de montrer comment utiliser au mieux ces outils en pratique et, en particulier, dans la pratique du chercheur et de l'ingénieur.

Utilisation du manuel

Ce livre s'adresse aux étudiants en deuxième année de licence ou au-delà, aux ingénieurs et aux chercheurs. Pour en tirer profit, il est utile d'avoir déjà suivi un cours de Python et de posséder des bases en analyse et en algèbre.

Le manuel peut être utilisé pour un cours d'un semestre composé de séances de cours, de travaux dirigés et de travaux pratiques. Cela constitue le contenu pour un module d'environ 10 à 15 séances de 3 heures, ce qui représente un total compris entre 30 et 50 heures d'enseignement.

Méthode pédagogique

Une des sources d'inspiration initiales de ce texte est le livre de Cleve Moler [104]. Ce livre s'appuyait lui-même sur une généalogie de livres dont le plus ancien est [46], un livre pionnier mêlant les méthodes, les algorithmes et les applications. Traduire en langage Python l'esprit de cette introduction au calcul numérique est rapidement apparu une approche incomplète dans la mesure où ces ouvrages laissent, d'une part, peu de place à la théorie et, d'autre part, aucune aux démonstrations. Puisque les démonstrations sont au cœur des mathématiques, nous avons complété le texte par des définitions et des théorèmes, associés à des preuves. L'objectif est de fournir au lecteur non seulement des méthodes applicables, des scripts Python fonctionnels, mais aussi une compréhension détaillée des calculs.

Dans le but de pouvoir concentrer la première lecture sur les concepts et non pas sur les détails, les démonstrations sont généralement présentées à travers les exemples

Nombre d'étoiles	Difficulté
Aucune	Mise en œuvre directe du cours
*	Un argument est requis
**	Plusieurs arguments sont requis
***	Plusieurs raisonnements se combinent

Table 1 – Difficultés des exercices.

ou détaillées sous la forme d'exercices.

Les exercices qui accompagnent la partie théorique du cours constituent une partie indispensable de l'apprentissage. Cela est une conséquence du choix de déplacer une partie de l'analyse de la section principale vers les exercices, dans le but de concentrer la présentation principale sur l'essentiel. Ces exercices sont triés par ordre de difficulté croissante, en fonction des critères indiqués dans la table 1.

Certains lecteurs trouvent que les démonstrations sont parfois trop détaillées. La qualité d'une démonstration tient essentiellement dans la qualité de son argumentation. En pratique, ce niveau de détail est bienvenu pour d'autres lecteurs, car cela leur permet de comprendre un point qui leur a échappé.

Partout où le soupçon d'une astuce mathématique se laissait entrevoir, nous avons préféré l'analyse rendant à la démonstration son caractère logique et cohérent avec l'objectif poursuivi. C'est la raison pour laquelle, parmi plusieurs démonstrations possibles d'un théorème, nous avons toujours préféré la plus riche du point de vue pédagogique, même si ce n'est pas la plus courte. Il s'ensuit des démonstrations aussi rigoureuses que possible, mais occasionnellement longues : le lecteur nous le pardonnera plus volontiers (nous l'espérons), pourvu que cette démonstration ait pu révéler des mécanismes intéressants.

Une partie significative de ce texte est consacrée au conditionnement des problèmes que nous allons traiter. En effet, bien qu'il occupe une place importante dans le calcul impliquant des nombres à virgule flottante, le conditionnement n'est pas un concept lié uniquement au calcul numérique, mais un concept mathématique associé à l'idée plus générale de changement de la solution lorsque les données sont perturbées. Cette idée, associée au concept de *problème bien posé*, ne peut pas être ignorée lorsqu'on utilise des méthodes numériques.

À chaque fois que cela est possible et pertinent, nous illustrons le concept mathématique par une figure, que celle-ci soit créée par une librairie de dessin vectoriel ou qu'elle soit créée en Python. L'exercice, pour le lecteur, consiste alors à observer la figure en détail pour faire le lien entre la figure et la définition ou le théorème. C'est un exercice que j'ai souvent demandé à mes étudiants, en précisant la grille de lecture d'un tel graphique : l'axe des abscisses, l'axe des ordonnées, le contenu du graphique et le lien avec le concept mathématique.

Dans certains contextes, la bibliographie francophone a des traditions dont certaines sont appropriées et d'autres non. Lorsque le présent manuel dévie de la tradition, nous expliquerons notre choix de manière explicite. C'est, par exemple, le cas pour la formule de Taylor page 38 ainsi que pour le théorème de la valeur intermédiaire page 363.

Certaines sections présentent des concepts plus avancés et dont la compréhension peut être difficile lors d'une première lecture, en fonction du niveau du lecteur. Nous avons indiqué ces chapitres par un astérisque (*) dans le titre. C'est le cas par exemple

pour le chapitre 6.6, qui peut être passée lors d'une première lecture. Au contraire, le lecteur désireux d'approfondir ses connaissances pourra utiliser ces indications pour identifier les chapitres les plus avancés. Par exemple, cela peut être utile pour un enseignant qui souhaiterait dispenser ce cours à un niveau supérieur.

Valeurs numériques

En langue française, le séparateur décimal est la virgule « , », ce qui est différent de la langue anglaise dans laquelle le séparateur décimal est le point « . ». Nous allons voir par la suite que le langage Python utilise la convention anglaise pour le séparateur décimal et utilise la virgule pour d'autres objectifs. Dans le contexte de ce livre, à la croisée des mathématiques et de l'informatique, dans le but de rendre le texte cohérent, nous avons adopté la convention anglaise pour le séparateur décimal, c'est-à-dire le point.

Un livre de calcul scientifique ne contenant aucune valeur numérique serait une absurdité. En effet, nous allons analyser en détail pourquoi les nombres flottants avec lesquels la plupart des utilisateurs de Python réalisent des calculs ont approximativement 16 chiffres significatifs. Toutefois, écrire toutes les valeurs numériques avec une précision aussi importante n'apporte pas nécessairement d'information pertinente.

Pour cette raison, nous arrondissons tous les résultats numériques au plus proche, avec 4 chiffres significatifs. Cela signifie que, lorsque le nombre mathématique est « 1.23456 », nous écrirons dans le texte « 1.235 ». De même, le nombre « 0.001234 » possède 4 chiffres significatifs, car les zéros devant le premier chiffre non nul ne comptent pas. Nous utiliserons occasionnellement la notation scientifique « 1.234×10^{-3} ».

Une précision importante est que nous présentons 4 chiffres significatifs y compris dans la plupart les résultats des sessions Python (mais Python utilise davantage de chiffres significatifs dans ses calculs). C'est une des raisons pour lesquelles, si le lecteur reproduit les scripts Python présentés dans ce texte, les valeurs numériques peuvent être légèrement différentes. Une autre raison fréquente est que les sessions présentées dans ce texte ont été exécutées sur différents systèmes (Linux et Windows), ce qui peut produire des résultats légèrement différents. Ces différences sont la plupart du temps sans conséquence. Plus de détails sur ce sujet seront présentés dans le chapitre 4 consacré aux nombres flottants.

Thèmes

Les différentes méthodes que nous allons détailler dans ce livre forment un réseau, de telle sorte que certaines méthodes s'appuient les unes sur les autres. La figure 1 présente certains de ces liens. Tentons de les éclaircir.

D'une manière générale, deux approches mathématiques se font face : l'algèbre et l'analyse. Dans notre contexte, deux sujets sont plus difficiles, plus fondamentaux ou plus importants que d'autres, car ils sont à la base de nombreuses autres techniques : il s'agit des systèmes d'équations linéaires (algèbre) et des équations différentielles ordinaires (analyse). D'un côté, les systèmes d'équations linéaires sont un thème essentiel en algèbre linéaire et ouvrent la porte à de nombreuses applications ainsi qu'à d'autres sujets, en particulier en algèbre linéaire numérique. D'un autre côté, les équations dif-

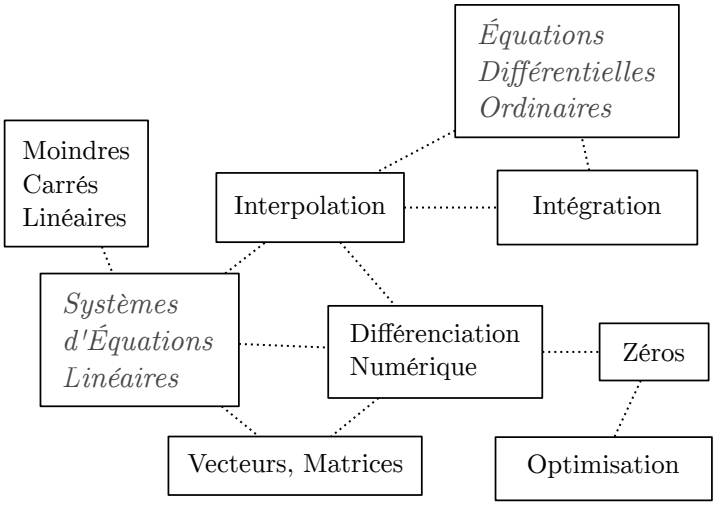

FIGURE 1 – Liens entre les méthodes numériques.

férentielles ordinaires sont à la base de nombreux modèles physiques et ouvrent la porte aux équations aux dérivées partielles.

La figure 1 montre bien que certains thèmes sont en réalité à la croisée de l'algèbre et de l'analyse. Dans le contexte de l'interpolation par exemple, représenter une fonction sur une base de fonctions donnée nécessite de considérer un espace vectoriel de dimension infinie. Puisque ce n'est pas pratique, nous ne représentons souvent que la projection de cette fonction sur un espace vectoriel de dimension finie, pour lequel l'algèbre fournit les outils de calcul appropriés. Nous laissons le soin au lecteur d'établir de nouveaux liens entre ces méthodes.

Scripts d'accompagnement

Les librairies `NumPy` et `SciPy` sont d'excellentes librairies de calcul scientifique et nous les utiliserons tout au long de cet ouvrage. Toutefois, dans certaines circonstances, nous pensons que les fonctionnalités dont nous avons besoin pour un thème précis sont mieux illustrées avec des algorithmes simplifiés, moins robustes, mais dont le code source est plus explicite.

C'est la raison pour laquelle nous avons développé une collection de scripts Python pédagogiques accompagnant le cours. Ces scripts sont disponibles sur le dépôt `github.com/mbaudin47/menum_code`. La table 2 présente la liste des modules. Ils sont disponibles dans le répertoire `Scripts-Eleves/Py3` du dépôt. Pour les utiliser, il est nécessaire de mettre à jour la variable `PYTHONPATH`, de telle sorte que l'environnement Python puisse connaître le nom du répertoire contenant les scripts. Configurer cette variable peut dépendre du système d'exploitation. Sur Linux, par exemple, on peut exécuter la ligne suivante dans le terminal :

```
export PYTHONPATH="/home/monlogin/Scripts-Eleves/Py3:$PYTHONPATH"
```

Ces modules pédagogiques sont utiles pour l'étude des algorithmes, mais je ne recommande à personne de les utiliser pour des études d'ingénierie ou bien pour

`floats.py`	Nombres à virgule flottante
`fzero.py`	Équations non linéaires
`interp.py`	Interpolation
`leastsq.py`	Moindres carrés linéaires
`linalg.py`	Systèmes d'équations linéaires
`numdiff.py`	Différentiation numérique
`odes.py`	Équations différentielles ordinaires
`optim.py`	Optimisation
`quadrature.py`	Intégration

Table 2 – Modules fournis avec le manuel.

la recherche : leur objectif n'est ni la performance, ni la précision, ni la souplesse d'utilisation, ni la robustesse. En effet, il existe une différence importante entre un algorithme et son implémentation (nous reviendrons sur ce thème dans la section 5.9). Toutefois, ces modules sont documentés et testés.

Pour chaque chapitre, les scripts sont disponibles dans le sous-répertoire `Scripts`. Pour le chapitre intégration, par exemple, les scripts sont disponibles dans le répertoire `Integration/Scripts`.

Pour les scripts les plus simples, nous fournissons parfois un script « squelette », contenant une base de travail que l'élève peut compléter durant les séances de travaux pratiques en Python. Dans ce script, le travail consiste à remplacer les instructions `TODO` par du code valide. L'objectif est de ne pas avoir à débuter par un script vide. Le script suivant[1], par exemple, présente un modèle de script pour l'intégration numérique. Nous indiquons dans une note de bas de page le nom du fichier correspondant au script présenté dans l'exemple ou dans la figure correspondante.

```
# Import des modules
TODO

def myfunB(x):
    y = TODO
    return y

Q, fcount = adaptsim_gui(myfunB, 0.0, 1.0)
```

Le script suivant[2] présente la solution du problème.

```
import numpy as np
from quadrature import adaptsim

def myfunB(x):
    y = 1.0 / np.sqrt(1.0 + x ** 4)
    return y

Q, fcount = adaptsim(myfunB, 0.0, 1.0)
```

Occasionnellement, nous distinguons une fonction mathématique de son implémentation en Python. Dans ce cas, nous employons un changement de fonte pour

[1] Script Python : `Integration/Scripts/Py3/exemples-squelette.py`.
[2] Script Python : `Integration/Scripts/Py3/exemples.py`.

$\mathbb{R}$	Nombres réels, par ex. $x = 1.234$
$\mathbb{R}^\star$	Nombre réels non nuls : $\{x \in \mathbb{R},\ x \neq 0\}$
$\mathbb{C}$	Nombres complexes, par ex. $z = 1 + 2i$
$\mathbb{N}$	Entiers naturels, par ex. $i = 1, 2, 3, \ldots$
$\mathbb{Z}$	Entiers relatifs, par ex. $m = \ldots, -2, -1, 0, 1, 2, \ldots$
$\mathbb{Z}^\star$	Entiers relatifs non nuls : $\{k \in \mathbb{Z}$ tels que $k \neq 0\}$
$\mathrm{Re}(z)$	La partie réelle du nombre complexe $z \in \mathbb{C}$
$\mathrm{Im}(z)$	La partie imaginaire du nombre complexe $z \in \mathbb{C}$
x, y, z	Des nombres réels
i, j, k	Des entiers
$[a, b]$	L'ensemble : $\{x \in \mathbb{R} \mid a \leq x \leq b\}$
(a, b)	L'ensemble : $\{x \in \mathbb{R} \mid a < x < b\}$
`x`	Variable Python

Table 3 – Notations.

α	A	Alpha	ι	I	Iota	ρ	P	Rho
β	B	Beta	κ	K	Kappa	σ	Σ	Sigma
γ	Γ	Gamma	λ	Λ	Lambda	τ	T	Tau
δ	Δ	Delta	μ	M	Mu	υ	Y	Upsilon
ϵ	E	Epsilon	ν	N	Nu	ϕ	Φ	Phi
ζ	Z	Zeta	ξ	Ξ	Xi	χ	X	Chi
η	H	Eta	o	O	Omicron	ψ	Ψ	Psi
θ	Θ	Theta	π	Π	Pi	ω	Ω	Omega

Table 4 – Lettres grecques.

signifier cette distinction. Par exemple, nous notons sin la fonction mathématique trigonométrique et `sin` son implémentation en Python.

Notations

Nous utiliserons les notations présentées dans la table 3. La table 4 présente les lettres grecques utilisées dans l'ouvrage. Dans le contexte de l'algèbre linéaire, les conventions présentées dans la table 5 sont adoptées. Elles correspondent à celles employées dans [70] page 2 à une exception : nous notons en caractères gras les vecteurs ainsi que le fait [132]. Ce livre a été préparé avec LaTeX[3].

Remerciements

J'adresse des remerciements tout particuliers à Jean-Marc Martinez qui m'a fait confiance pour commencer à enseigner ce type de méthodes et sans qui ce livre n'existerait probablement pas. Ses commentaires et correction sur ce manuel ont étés très bénéfiques. Mes remerciements vont à Bernard Hugueney qui m'a encouragé à approfondir ce sujet. Je remercie les élèves de l'Institut supérieur d'électronique de Paris

[3] https ://www.latex-project.org/

α	Scalaire (réel ou complexe) : lettre grecque
$\mathbf{x}$	Vecteur : lettre latine, minuscule, en gras
$\langle \mathbf{x}, \mathbf{y} \rangle$	Produit scalaire des vecteurs $\mathbf{x}$ et $\mathbf{y}$
$\mathbb{R}^n$	Ensemble des vecteurs réels de dimension n
$\mathbb{R}^{m \times n}$	Ensemble des matrices réelles à m lignes et n colonnes
$\mathbf{x}$	Vecteur : lettre latine, minuscule, en gras
diag	Matrice diagonale
$\mathbf{I}$	Matrice identité
rang	Rang d'une matrice
$\|\mathbf{x}\|$	Norme du vecteur $\mathbf{x}$
P	Matrice de permutation
Q	Matrice orthogonale
L	Matrice triangulaire inférieure
U	Matrice triangulaire supérieure
A^{-1}	Inverse de A
$\kappa(A)$	Conditionnement de la matrice A
$A^\dagger$	Pseudo-inverse de A

TABLE 5 – Notations pour l'algèbre linéaire.

(ISEP) qui, par leurs questions et leurs commentaires, m'ont stimulé dans la rédaction de ce document. Mes remerciements s'adressent également aux enseignants qui m'ont accompagné sur le module de méthodes numériques, en particulier Omar Al Hammal. Je remercie le Dr. Jean-Pierre de Mondenard pour sa relecture du chapitre consacré aux applications et en particulier à la prévention de la commotion cérébrale. Je souhaite remercier Régis Lebrun pour ses suggestions et corrections. Je tiens à remercier Vincent Chabridon pour ses encouragements et ses corrections. Je remercie enfin mon épouse Vi-An Baudin pour ses relectures, commentaires et corrections détaillées.

J'invite le lecteur à me transmettre ses suggestions à

`methodes.numeriques.python@gmail.com`.

Michaël Baudin,
Août 2022

Table des matières

Première partie

Prolégomènes au calcul scientifique

Chapitre 1

À quoi sert le calcul numérique ?

Les objectifs de ce chapitre sont multiples. Premièrement, ce chapitre a pour but de définir le calcul numérique et sa position par rapport aux autres outils scientifiques qui lui sont souvent associés. Deuxièmement, il tente de présenter le calcul numérique en action, à travers ses applications et ses enjeux. Enfin, il tente de montrer l'importance que cette discipline, à la croisée des mathématiques appliquées, de l'informatique et de la plupart des sciences applicatives (par exemple la physique et la chimie), a prise dans le monde contemporain.

Pour mieux cerner les applications pratiques de la simulation numérique, on peut énumérer quelques exemples de problèmes d'ingénierie classiques. Il s'agit parfois de réduire le poids (kg), d'assurer la fiabilité ou de réduire le coût (€). Presque toutes les industries ont recours au calcul numérique : nous allons voir quelques-unes de ces applications.

1.1 Prévention de la commotion cérébrale chez les cyclistes

La conception des casques de cyclisme peut faire appel à la simulation numérique. Comme nous allons le voir, ce problème d'ingénierie révèle beaucoup de caractéristiques de la pratique du calcul numérique. En 2006, S. Kleiven a utilisé un modèle numérique de la masse cérébrale permettant de prévoir, dans le volume crânien, la pression en fonction de l'espace et du temps lorsqu'un crâne est soumis à un impact [83]. Ce modèle est fondé sur la méthode des éléments finis, une méthode numérique avancée souvent utilisée en mécanique. Dans la figure 1.1, le volume de la tête est décomposé en différentes sous-parties. En commençant par les parties les plus externes, ces volumes sont la boîte crânienne, les lobes, le cervelet et la moelle spinale[1]. Puisque le comportement mécanique de chaque partie est spécifique (par exemple, la boîte crânienne se déforme moins facilement que la matière grise), il est nécessaire de spécifier localement les propriétés mécaniques de chaque partie. Dans ce contexte,

[1] En ancienne nomenclature, la moelle spinale est nommée moelle épinière et le fluide cérébrospinal est nommé fluide céphalorachidien.

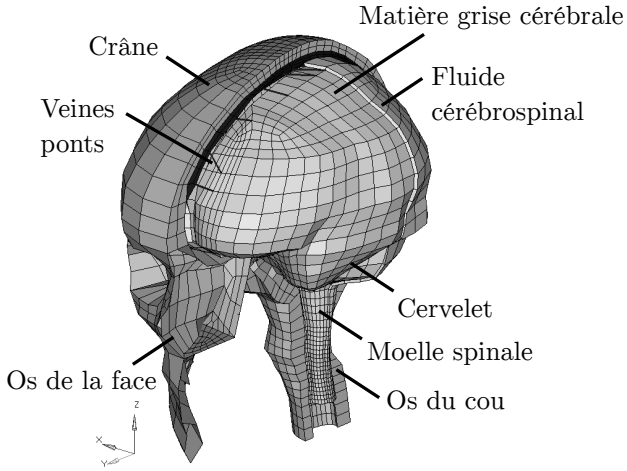

FIGURE 1.1 – Discrétisation d'une tête humaine [84]. Le crâne, le liquide cérébrospinal et la matière grise. Illustration reproduite avec l'aimable autorisation de S. Kleiven.

on utilise le module d'Young qui caractérise la relation entre la déformation et la contrainte exercée [85]. Chaque volume est associé à un maillage spécifique, qui est une discrétisation de sa géométrie.

Ces travaux faisaient suite à des recherches initiées par le neurochirurgien suédois Hans von Holst à la fin des années 1990 pour identifier des améliorations dans les casques de protection individuels. Il a alors contacté l'École royale polytechnique (KTH) de Stokholm pour initier un programme de recherche sur ce thème, ce qui a mené Peter Halldin, un étudiant de cet institut, à commencer une thèse en biomécanique. C'est dans ce contexte que le travail de recherche de S. Kleiven débutera quelques années plus tard [85] et mènera à l'article de 2006 que nous sommes en train d'analyser.

Les impacts pris en compte dans l'étude sont caractérisés par leur direction frontale, occipitale (à l'arrière du crâne) ou latérale et par leur intensité. L'objectif final de ce travail était en particulier de pouvoir prévoir l'apparition de la commotion cérébrale liée à l'impact sur la tête et pouvant mener, dans certains cas, à des séquelles temporaires, définitives, ou à la mort.

Le chercheur souhaitait confronter les résultats du modèle numérique à la réalité. Il était évidemment hors de question de projeter des cobayes humains contre des obstacles pour observer l'apparition d'une commotion cérébrale. Pour résoudre cette difficulté, l'auteur a utilisé dans [83] des données issues de mesures expérimentales sur des cadavres. Toutefois, ces données doivent être corrigées pour prendre en compte le changement potentiel de rigidité de la matière cérébrale après la mort. De plus, les mesures pourraient être affectées par la présence d'air dans le fluide céphalo-rachidien, ce qui rend difficile les comparaisons avec la simulation numérique.

Dans [84, 82], l'auteur a utilisé des images de football américain captées par la télévision pour observer si on pouvait reproduire par la simulation les traumatismes observés sur les joueurs. Il a utilisé des indicateurs comme l'accélération angulaire, l'accélération de translation et la puissance d'impact pour observer si ces indicateurs sont de bons prédicteurs de la pression intracrânienne et des déformations associées à

la blessure. Les résultats montrent une corrélation statistique entre les déformations, les contraintes et l'apparition de blessures dans certaines zones du cerveau, et en particulier dans la matière grise.

En analysant les mécanismes générant des déformations et des contraintes dans le cerveau, les chercheurs ont compris que celui-ci est bien plus sensible à des rotations qu'à des déplacements linéaires. Il s'avère que, bien que la plupart des casques de cyclisme soient correctement testés en laboratoire par des organismes de certification vis-à-vis d'un impact linéaire, ce n'est pas ce type d'impact qui peut générer le plus facilement une commotion : il y a souvent plus de dommages lors d'un impact avec un angle entre le casque et l'obstacle, à cause de la rotation qui s'ensuit.

La boîte crânienne protège le cerveau d'intrusions extérieures. Du côté intérieur, si le cerveau était directement en relation avec la boîte crânienne, le moindre choc pourrait endommager le cerveau, qui est contraint à rester à l'intérieur de la boîte. Un des rôles du fluide entre le cerveau et la boîte crânienne (liquide céphalo-rachidien) est de limiter ces mouvements potentiellement brutaux. De plus, il peut permettre de limiter l'impact d'une accélération de rotation. L'idée des chercheurs a alors été de reproduire ce mécanisme naturel grâce à une couche de plastique mobile à l'intérieur du casque. Cette couche plastique peut se déplacer d'environ 10 à 15 millimètres sur la surface intérieure du casque. Ainsi, lors d'un choc avec un angle, la vitesse de rotation du casque n'est pas immédiatement transmise à la boîte crânienne, mais elle est amortie par l'interface plastique mobile, ce qui atténue la rotation. Pour valider leur approche, le modèle numérique cérébral a été utilisé : il montre que la technique peut permettre de limiter les déformations et les contraintes dans le cerveau.

Depuis, cette technologie a été commercialisée par une société [103] et de nombreux casques de cyclistes en sont équipés. Bien sûr, la technologie a également été testée en laboratoire par des essais mécaniques, mais il n'est pas si simple de reproduire en laboratoire la dynamique de la masse cérébrale à l'intérieur de la boîte crânienne. Cet exemple révèle l'utilisation conjointe de méthodes numériques et d'essais en laboratoire dans l'objectif d'améliorer la sécurité.

Dans la section suivante, nous allons élargir la discussion pour analyser comment les méthodes numériques et les essais en laboratoire se positionnent dans le panorama des méthodes scientifiques.

1.2 Réel et modèle

Lorsque l'on cherche à comprendre le comportement d'un système réel, on peut soit procéder à des expériences, soit créer un modèle du système, comme le montre la figure 1.2. Les expériences, par exemple en laboratoire ou bien sur le terrain, mènent à des résultats expérimentaux, souvent entachés d'erreurs de mesures. D'un autre côté, le modèle peut être traité soit par une approche théorique, soit par simulation. L'approche par simulation, qui est le sujet de ce cours, mène à des résultats de simulation, qui sont fréquemment associés à des erreurs d'arrondis et, parfois, à des erreurs d'approximation. Une difficulté importante est que la plupart des modèles théoriques n'ont pas de solution explicite. On peut alors utiliser des modèles théoriques approchés menant à des prédictions théoriques. Les trois types de résultats (expérimentaux, théoriques ou issus de simulations numériques) peuvent être comparés, ce qui peut mener à améliorer en retour ces trois approches.

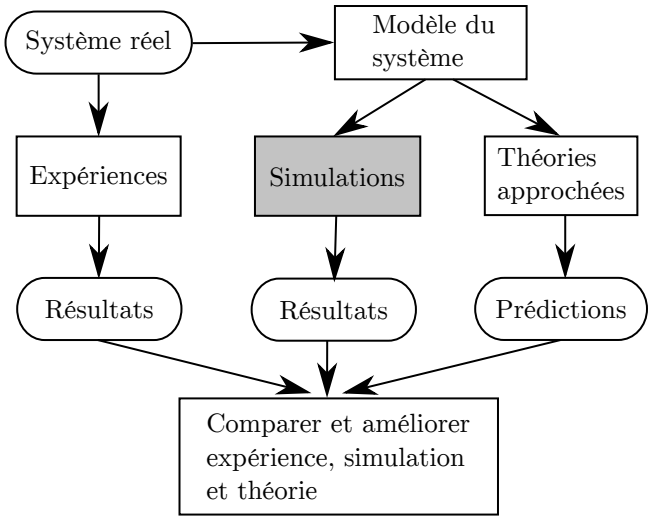

FIGURE 1.2 – Position des expériences, de la simulation numérique et des théories approchées [30].

L'approche expérimentale est indispensable dans la mesure où c'est elle qui possède, en première instance, la plus grande proximité avec la réalité. Toutefois, c'est une approche coûteuse en argent et aussi, nous allons le voir, en temps. De plus, les moyens expérimentaux nécessaires pour pouvoir réaliser les mesures dans de bonnes conditions de sécurité mènent parfois à s'éloigner du système réel.

L'approche par simulation numérique est, en pratique, plus aisée (nous espérons convaincre le lecteur de cela à l'issue cet ouvrage) et souvent performante, notamment grâce aux spectaculaires progrès de l'informatique, à la fois sur le plan logiciel et matériel. Toutefois, elle peut s'avérer complexe à mettre en œuvre, par exemple lorsque le coût de calcul (en termes de performance) s'avère élevé. Actuellement, la relative démocratisation du calcul parallèle (distribué) et des supercalculateurs tend à modérer ce coût. Surtout, le coût de développement des logiciels de calcul numérique peut se traduire soit par des coûts internes de développement et de formation des développeurs, soit par des coûts d'achat de licence. Enfin, la simulation numérique est toujours associée à des approximations, que le présent cours va présenter en détail, mais qui sont potentiellement difficiles à contrôler. L'approche théorique fondée sur les méthodes exactes, analytiques, est souvent précise. Une limitation déterminante de cette approche est que bon nombre de systèmes d'équations n'ont pas de solution exacte connue. De manière alternative, on peut, dans certains cas, utiliser des systèmes de calcul symbolique, mais ceux-ci peuvent s'avérer moins performants.

Bien sûr, la plupart des ingénieurs et chercheurs utilisent les trois approches lorsque cela est possible et il n'y a pas une approche qui serait, en toute généralité, supérieure aux autres, comme nous allons le voir en détail par la suite. Dans la section suivante, nous présentons un exemple d'essai expérimental classique : l'essai en soufflerie.

1.3 Exemple d'essais en soufflerie : application aux jantes de cycle

Une des forces limitant la progression de l'ensemble cycliste-bicyclette dans l'air est la résistance due à la force de traînée aérodynamique [56, 18]. Celle-ci dépend en particulier de la surface frontale de l'ensemble cycliste-bicyclette, du carré de la vitesse (incluant la vitesse de la bicyclette et du vent), de la densité de l'air et d'un coefficient nommé *coefficient de traînée*. Ce coefficient dépend de plusieurs facteurs, dont la forme de l'ensemble cycliste-bicyclette. D'après Jean-Pierre Mercat, responsable de recherche MAVIC, un cycliste roulant à 50 km/h sur le plat développe 90% de sa puissance pour vaincre la force de traînée [68].

C'est la raison pour laquelle la société MAVIC utilise des techniques issues de l'aéronautique dans le but de créer des jantes de cycle plus performantes comme la Cosmic CXR. Parmi ces techniques, une des méthodes consiste à concevoir la forme de la jante de telle sorte que l'ensemble pneu-jante suive un profil aérodynamique minimisant le coefficient de traînée. Dans le but de minimiser la force de traînée, on utilise des méthodes numériques issues de la mécanique des fluides. Ce type de profil est parfois nommé profil « NACA », du nom du centre de recherche des USA chargé d'élaborer des profils d'aile d'avion à partir des années 1920. En effet, le *National Advisory Committee for Aeronautics* (NACA), créé en 1915, était un organisme de recherche dont le but était de conduire des recherches dans le domaine de l'aéronautique. Cet organisme est ensuite devenu en 1958 la *National Aeronautics and Space Administration* (NASA) pour inclure les recherche dans le domaine de l'espace [107].

Une fois le profil dessiné, la maquette doit ensuite être testée pour observer ses performances en situation réelle. Pour cela, la société MAVIC a collaboré avec le CMEFE-HEPIA[2], un centre de recherche actif dans les mesures aérodynamiques localisé à Genève en Suisse. La collaboration a duré 3 ans, de 2009 à 2012, en particulier pour utiliser la soufflerie HEPIA et leurs compétences dans les mesures associées [68]. La figure 1.3 présente un essai de jantes MAVIC dans la soufflerie HEPIA. On y distingue le cycliste sur sa machine placé dans un tunnel équipé de ventilateurs, une situation qui simule l'air circulant autour du cycliste lorsqu'il se déplace : ici, le cycliste est immobile et c'est l'air qui s'écoule autour de lui. On y distingue au premier plan un tube métallique, nommé tube de Pitot, dont le but est de mesurer la vitesse de l'air. D'après Jean-Pierre Mercat, les essais ont étés menés durant plus de 500 heures [68].

Une des difficultés associées à ces essais en soufflerie est leur coût. Puisque les soufflerie coûtent cher, elles sont peu nombreuses et seuls certains laboratoires, comme le CMEFE-HEPIA, en disposent. Il n'est pas facile d'indiquer précisément le prix de l'utilisation d'une telle soufflerie, car les investissements sont généralement associés à la fois à l'achat du matériel, mais également au considérable investissement humain associé à ce type d'équipement nécessitant de grandes compétences expérimentales. Frédéric Grappe ([56], page 9) indique la somme d'environ 100 000 Francs par jour en 2000, un montant équivalent à environ 20 000 €par jour en 2021. Toutefois, en comparaison d'autres essais, ce coût peut paraître acceptable, voire insignifiant, ce que nous allons voir dans la section suivante.

[2] `http://www.cmefe.ch/`

FIGURE 1.3 – Essais en soufflerie de jantes MAVIC. Avec l'aimable autorisation de la soufflerie HEPIA Genève, de MAVIC et de `cinemargot.com`.

1.4 Essais difficiles, coûteux ou impossibles

Dans certaines situations, les essais sont très coûteux, voire impossibles à réaliser. Le but de cette section est de présenter quelques exemples d'essais expérimentaux de ce type, les difficultés associées et leur position par rapport à la simulation numérique.

Les essais de chocs de voiture ou *crash-test* sont des essais destructifs ayant pour objectif d'observer et mesurer les effets d'un choc sur un véhicule [59]. Aux États-Unis, le programme *New Car Assessment Program* (ou NCAP) est créé en 1979 dans le but de créer des automobiles plus sûres pour le véhicule lui-même et pour ses occupants. Ce test consistait en un choc frontal à 35 mph, soit environ 56 km/h. Dans le programme européen EURO-NCAP de 2020 les véhicules sont projetés de face contre un obstacle fixe ou mobile. Un autre test consiste à projeter un obstacle mobile sur le côté du véhicule à la vitesse de 60 km/h. La dynamique de déformation du véhicule se déroule en une fraction de seconde. Ces essais ont l'inconvénient de coûter plutôt cher, et c'est une des raisons pour laquelle la simulation numérique par la méthode des éléments finis est utilisée [59]. La simulation permet en effet de comprendre la dynamique plus en détail, ainsi que de réduire le nombre d'essais nécessaires.

On peut citer d'autres exemples d'essais coûteux. Dans le cadre du programme Apollo, le centre de recherche de la NASA, à Langley (en Virginie, aux États-Unis), a été utilisé pour tester l'atterrissage lunaire d'un véhicule. Ce site expérimental a ensuite été utilisé pour tester la robustesse d'autres équipements. Par exemple, en 1984, trois avions de transport Boeing 707 ont étés testés dans le cadre d'essais de chute verticale [76].

Entre les phases d'essais et de commercialisation, l'étape de qualification d'un

produit peut être utilisée pour observer le comportement de l'objet en conditions réelles. Durant cette phase, l'objet à qualifier est équipé de capteurs, ce qui peut permettre de détecter et de corriger des problèmes qui n'ont pas été identifiés lors des essais préalables. Une telle situation a eu lieu dans le cadre du programme de lancement de satellites via le lanceur Ariane 5. En 1996, le vol de qualification du vol 501 d'Ariane 5 est un échec et 4 sondes sont perdues, d'une valeur de 370 millions de dollars. De plus, le programme Ariane 5 est retardé d'un an [90]. La cause du problème est un dysfonctionnement informatique.

Lorsque les essais sont impossibles, il faut trouver des alternatives. C'est le cas de tous les essais associés à des défaillances de systèmes critiques. De toute évidence, il n'est pas question de créer un accident de perte de réfrigérant primaire pour étudier le comportement d'une centrale nucléaire de production d'électricité en situation accidentelle. C'est pourquoi, dans le but de caractériser le comportement de la première barrière de confinement de la radioactivité, des essais sur la gaine du crayon combustible sont réalisés [64].

On peut avoir le sentiment *a priori* que ces essais difficiles ou impossibles ne sont qu'une question de coût et qu'un investissement supplémentaire pourrait résoudre les difficultés. En fait, dans certains cas, la durée de l'essai est beaucoup plus longue que l'on pourrait le croire et cette durée ne peut pas être réduite facilement. Par exemple, les deux sondes *Voyager 1* et *Voyager 2*, lancées en août et septembre 1977, ne survolent Saturne qu'en novembre 1980 et août 1981 [65]. La sonde *Voyager 2* ne s'approche de Neptune qu'en 1989, c'est-à-dire 12 ans après son lancement.

Après avoir réalisé des essais militaires de bombes nucléaires en Algérie, puis en Polynésie, la France a arrêté définitivement en 1996 d'utiliser ce type d'essais [21]. Dans le but de maintenir la performance et la sûreté de ces armes, le Commissariat à l'Énergie Atomique (CEA) a engagé un programme appelé « Simulation » [110]. Cela a mené à l'utilisation d'une famille de supercalculateurs parallèles à Bruyères-Le-Chatel, les machines « TERA », un nom issu de teraflops, l'unité de mesure du nombre d'opérations par seconde d'un calculateur. De plus, des équipements expérimentaux comme le Laser MégaJoule ont étés créés. Ce laser est installé dans le Centre d'Études Scientifiques et Technique d'Aquitaine (CESTA), un site du CEA localisé en Gironde.

1.5 Conclusion

Dans ce chapitre, nous avons vu le positionnement de la simulation numérique et son utilité en complément des indispensables essais expérimentaux. L'objectif de ce chapitre est de montrer au lecteur qui n'aurait utilisé que des méthodes de résolution exactes ce que la simulation numérique peut apporter dans certaines applications. Malheureusement, la simulation n'est pas la solution « miracle » et beaucoup de défis théoriques et pratiques restent à résoudre. Une difficulté est le coût de simulation en temps de calcul. Par exemple, il n'est pas rare qu'une simulation en dynamique des fluides, menée sur des milliers de processeurs en parallèle, dure de plusieurs heures à plusieurs jours, voire plusieurs semaines. Un autre problème est l'assurance dans la qualité de la prédiction des simulations numériques. Sur ce sujet, un des thèmes de recherche actuels est la validation des codes de calcul par comparaison avec les essais, un sujet faisant partie de la vérification et validation des codes (appelée « V&V ») [137].

1.6 Notes et références

Je remercie Jean-Pierre de Mondenard pour ses conseils dans la rédaction de la section 1.1, dans le contexte de la chute de Romain Bardet survenue le vendredi 11 septembre 2020 lors de la treizième étape du Tour de France menant de Châtel-Guyon au Puy Mary dans le Cantal.

Chapitre 2

Python

L'objectif de ce chapitre est de servir de base minimale à l'apprentissage du langage Python pour le calcul numérique. Au détour de certains thèmes, nous découvrirons des aspects spécifiques, parfois inattendus, du langage, en particulier pour les nombres flottants.

Ce chapitre ne saurait se substituer à un véritable cours sur ce sujet. C'est pourquoi nous conseillons au lecteur qui ne connaîtrait pas ce langage de consulter [112] et [34], deux références très populaires et librement disponibles sur internet. Puisque le niveau de programmation Python requis dans ce cours est plutôt minimal, seuls les premiers chapitres de ces ouvrages sont nécessaires.

Pour installer la librairie Python, il existe de nombreux moyens. Une distribution Python très populaire est la distribution Anaconda, disponible sur `www.anaconda.com` tant pour Windows, Linux ou bien Mac. Ce cours s'appuie sur Python 3 et nous utiliserons, de plus, les modules `NumPy`, `SciPy` et `Matplotlib`. Une bonne nouvelle pour les plus pressés d'entre nous est que la distribution Anaconda, quoique pesante en termes de taille de fichier, contient déjà les librairies dont nous avons besoin.

Sur les systèmes Linux, il est la plupart du temps facile d'installer Python en utilisant le logiciel de paquets, comme `Synaptic` sur Debian, par exemple. Souvent, le langage Python est déjà pré-installé sur ce système d'exploitation et il n'y a rien à faire de particulier. Toutefois, il peut être commode d'installer une version spécifique de Python. Sur Ubuntu par exemple, on peut installer Python 3.9 avec la commande :

```
sudo apt install python3.9
```

Une fois la distribution Python installée, il est parfois nécessaire d'installer des librairies particulières comme `NumPy` par exemple. Dans ce cas, on peut utiliser l'utilitaire `pip` qui permet de télécharger et installer des modules Python :

```
pip install numpy matplotlib scipy spyder
```

Les scripts présentés dans ce manuel sont testés avec Python 3.9, `NumPy` 1.23, `SciPy` 1.9 et `Matplotlib` 3.5.

Lorsqu'on écrit un code source en langage Python, il y a en fait peu de place pour le *style* de programmation. Plus précisément, le style désigne la manière d'indenter les blocs de code structurés, le lieu où on peut placer une ligne blanche, le choix de l'espace blanc ou de la tabulation, le nombre maximum de caractères par lignes, la manière de nommer les variables et beaucoup d'autres éléments. Ces éléments de style ne sont

pas imposés par le langage Python. Cependant, il est d'usage de suivre *strictement* les conventions du *Python Enhancement Proposal 8* ou *PEP8* [118], comme nous l'avons fait dans ce manuel. Pour faciliter ce travail, on peut utiliser le logiciel `black`[1] qui formate automatiquement le fichier passé en argument en suivant les conventions requises.

2.1 Variables

Après avoir ouvert l'interpréteur Python, nous obtenons une console commençant par trois chevrons `>>>`, que l'on nomme le *prompt*. La session Python suivante[2] présente quelques opérations arithmétiques élémentaires. Après le caractère « `#` », il y a des commentaires qui ne sont pas exécutés par Python. Le point décimal « `.` » sépare la partie entière et la partie fractionnaire. Insistons sur le fait, contrairement à la langue française, ce n'est pas la virgule qui est le séparateur décimal en Python (nous reviendrons sur ce sujet par la suite).

```
>>> x = 12.0   # Affecter une variable
>>> x          # Afficher la valeur de x
12.0
>>> y = 2 * x  # On multiplie x par 2
>>> y          # Afficher la valeur de y
24.0
>>> print(y)   # Afficher la valeur de y
24.0
>>> del y      # Effacer la variable y
>>> y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' is not defined
```

Le langage Python fournit plusieurs types de données, en particulier les nombres à virgule flottante (ou « `float` ») et les entiers (ou « `int` »). Pour écrire le nombre à virgule flottante 12, nous écrivons en général « `12.0` » dans le but de spécifier explicitement sa nature, bien que le langage permette également la syntaxe « `12.` ». Lorsqu'on écrit « `12` », Python considère que c'est un entier. Nous verrons par la suite que ces deux types de variables ont des comportements radicalement différents.

```
>>> x = 12.0
>>> type(x)
<class 'float'>
>>> x = 12
>>> type(x)
<class 'int'>
```

2.2 Listes et tuples

Dans cette section, nous introduisons deux structures de données complémentaires : les listes et les tuples.

[1] https ://pypi.org/project/black/
[2] Script Python : `Python/Scripts/Py3/langage.py`.

Indice `i`	0	1	2
Variable `a[i]`	`a[0]`	`a[1]`	`a[2]`
Valeur	`9.`	`5.`	`11.`

TABLE 2.1 – Indice, variable et valeur dans un tableau Python.

Pour définir une liste, on utilise les crochets « `[` » et « `]` », qui marquent le début et la fin de la liste. La virgule « `,` » sépare les éléments de la liste. Dans la session suivante[3], la liste `a` est de longueur 3.

```
>>> a = [9.0, 5.0, 11.0]
>>> type(a)
<type 'list'>
```

Pour la liste précédente, les indices commencent à `i=0` et terminent à `i=2`. La table 2.1 présente, pour chaque indice de 0 à 2, la variable correspondante ainsi que sa valeur. C'est la raison pour laquelle la valeur correspondant à l'indice `i=1` est la seconde valeur de la liste, comme on le voit dans la session suivante.

```
>>> a = [9.0, 5.0, 11.0]
>>> a[1]
5.0
```

Une exception de type `IndexError` est retournée par l'interpréteur lorsque l'indice est trop grand.

```
>>> a = [9.0, 5.0, 11.0]
>>> a[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Un tuple est une collection d'objets. La virgule « `,` » sépare les éléments du tuple. Les parenthèses « `(` » et « `)` » marquent le début et la fin du tuple. Une différence importante avec la liste est qu'on ne peut pas changer le contenu d'un `tuple`. Dans l'exemple suivant, la variable `x` est un `tuple` de longueur 3 contenant des entiers.

```
>>> x = (7, 12, 5)
>>> x
(7, 12, 5)
>>> type(x)
<type 'tuple'>
```

On peut se demander pourquoi utiliser un tuple, alors qu'une liste fournit les mêmes fonctionnalités, avec l'avantage supplémentaire que l'on peut modifier le contenu d'une liste (mais pas d'un tuple). Par exemple, si on doit créer un programme capable de gérer la liste des élèves d'une classe, mais que cette liste n'est pas connue à l'avance et peut changer au cours du temps, la liste semble, en effet, la structure de données la plus appropriée.

En pratique, il y a des situations où l'on souhaite justement empêcher de modifier le contenu d'une collection d'objets. Par exemple, supposons que l'on souhaite représenter les bits `0` et `1` par les caractères `"0"` et `"1"`. Puisqu'il n'existe aucun autre bit

[3]Script Python : `Python/Scripts/Py3/listes.py`.

possible, la structure appropriée est le tuple et une exception sera générée lorsqu'on tentera de modifier la collection de bits. En effet, c'est précisément dans le but de générer une exception dans cette situation que le développeur préférera utiliser un tuple plutôt qu'une liste.

En français, il y a une confusion possible, car le séparateur décimal est la virgule, alors que le séparateur décimal en Python est le point. C'est ce qui explique pourquoi la session suivante génère, de manière implicite, un tuple, au lieu du nombre à virgule flottante auquel l'élève pourrait s'attendre.

```
>>> x = 12,5
>>> x
(12, 5)
```

2.3 Conditions et boucles

Dans cette section, nous introduisons l'instruction `if` ainsi que les boucles fondées sur les instructions `while` et `for`.

L'instruction `if` permet de créer une structure conditionnelle. Le caractère « `:` » marque la fin de la condition du « `if` » et du « `else` ». À l'intérieur de chaque bloc, on doit indenter avec quatre espaces blancs. On n'utilise ni trois espaces blancs, ni cinq, ni une tabulation : exactement quatre espaces blancs. La condition suivante[4] affiche `"Negatif"` ou `"Positif ou nul"` selon le signe de `a`.

```
if a < 0.0:
    print("Negatif")
else:
    print("Positif ou nul")
```

En théorie, le langage Python n'oblige pas nécessairement à utiliser quatre espaces blancs pour indenter correctement le script : on pourrait parfaitement utiliser une tabulation. Pire, il est possible de mélanger les espaces blancs et les tabulations. Toutefois, le problème de la tabulation est que sa longueur graphique, c'est-à-dire la largeur physique que la tabulation occupe à l'écran, peut dépendre de l'encodage dans lequel le texte est sauvegardé. C'est la raison pour laquelle la plupart des développeurs utilisent l'espace blanc pour indenter leur code, car celui-ci est encodé de manière presque universelle de la même manière sur tous les systèmes.

Dans la session suivante, on affecte le nombre à virgule flottante 0 à la variable `a`. Puis, si `a` est négatif, on affiche « Negatif » et « Positif ou nul » sinon.

```
>>> a = 0.0
>>> if a < 0.0:
...     print("Negatif")
... else:
...     print("Positif ou nul")
...
Positif ou nul
```

Une boucle `while` s'exécute tant que la condition est vérifiée. Le caractère « `:` » marque la fin de la condition du « `while` ». À l'intérieur du bloc, on doit indenter avec quatre espaces blancs. Dans la session suivante[5], on utilise une boucle `while` pour

[4] Script Python : `Python/Scripts/Py3/condition.py`.
[5] Script Python : `Python/Scripts/Py3/boucles.py`.