UNIVERSITE DE NAMUR

Faculté des Sciences

Algorithmique et programmation au cours de mathématiques : une étude exploratoire avec Python.

Mémoire présenté pour l'obtention

du grade académique de master en mathématiques à finalité didactique

Estelle VAN HOUT

Aout 2020

Promotrice : Valérie Henry

Remerciements

En tout premier lieu, la personne que je tiens à remercier est ma promotrice, Madame Valérie Henry, qui m'a permis de réaliser un travail dans ma finalité, en didactique des mathématiques, et sur un sujet véritablement tout aussi intéressant qu'il en a l'air. Je tiens également à la remercier pour sa réactivité, son avis éclairé et ses conseils pertinents, mais aussi d'avoir cru en mes capacités.

Je souhaiterais remercier également Marvyn Gulina, pour sa gentillesse et son aide concernant la séquence de cours présentée dans ce mémoire, ainsi que le reste de l'équipe des enseignants et des assistants du département de mathématiques de l'Unamur pour leur accueil, leur encadrement et leurs précieux enseignements de ces dernières années.

Mes pensées vont également à mes amies Alice, Elodie, Marie et Léa qui m'ont accompagnée et épaulée toutes ces années, que ce soit dans les moments de joie ou les moments les plus difficiles. Ma gratitude va tout particulièrement à Alice pour les avis, commentaires et relectures qu'elle a pu faire sur ce travail, ainsi que pour son soutien et ses encouragements.

Un merci tout particulier à Mélanie, mon amie d'enfance, d'avoir toujours été là et de continuer à l'être encore pour longtemps.

Enfin, je remercie de tout mon cœur ma mère, mon père, mon frère, mes beaux-pères ainsi que Valentin de m'avoir entourée, encouragée, soutenue et même relevée lorsque les temps étaient durs. Sans eux je n'en serais pas là où j'en suis aujourd'hui, ils ont tout mon amour et mon éternelle reconnaissance.

Résumé

C'est il y a plus de 10 ans déjà, suite à la publication d'un rapport émis par la Commission Kahane en 2000, que la France a introduit des éléments d'algorithmique dans ses programmes d'enseignement. De notre côté, en Belgique, nous nous posons encore aujourd'hui de nombreuses questions sur les intérêts, les effets et les possibilités liées à l'introduction de l'algorithmique au sein du cours de mathématiques, notamment dans le secondaire supérieur. Dans ce mémoire, nous tâchons de répondre à quelques unes de ces questions en nous basant sur l'étude de la situation de la France, ainsi que l'identification théorique des liens entre algorithmique, programmation, mathématiques et TICE. Ainsi nous mettons en lumière, au travers de l'analyse à priori d'une séquence de cours, les différents phénomènes qui affectent le savoir, les élèves et leur apprentissage dans une situation mêlant les mathématiques, l'algorithmique et la programmation dans le langage de programmation Python.

Mots-clés : algorithmique, programmation, mathématiques, Python, enseignement, secondaire supérieur

Abstract

More than ten years ago, it has been decided in France to include algorithmic elements in its education programmes following the publication of a report from the Kahane Commission. As far as Belgium is concerned, we still wonder nowadays if the inclusion of algorithmic elements in the secondary education might have any interest, consequences, or potential. In the present dissertation we are trying to answer some of these questions by relying on the study of the situation in France, as well as on the theoretical identification of the links between algorithmics, programming, mathematics, and ICTE. By analysing a teaching sequence, we are trying to highlight the different phenomena which influence the knowledge, the students, and their learning in a situation in which mathematics, algorithmics and programming are mingled together in the Python programming.

Key words : algorithmics, programming, mathematics, Python, teaching, secondary education

Table des matières

Introduction

Dans ce mémoire en didactique des mathématiques, nous nous intéressons à l'algorithmique et à la programmation au cours de mathématiques. Plus précisément, nous cherchons à étudier les effets de l'introduction de l'algorithmique dans l'enseignement des mathématiques du secondaire supérieur, et ce en considérant le langage de programmation Python.

L'intérêt d'une telle étude réside dans le fait que ces domaines sont presque totalement absents du paysage actuel de l'enseignement secondaire en Belgique. Or, ce n'est pas le cas en dehors du pays. En effet, chez nos voisins français, des éléments d'algorithmique et de programmation ont été intégrés dans les programmes depuis une dizaine d'années déjà, autant au collège qu'au lycée. Par conséquent, il est intéressant de se demander pourquoi et comment les programmes français ont changé, mais aussi quels sont les impacts sur l'apprentissage des principaux intéressés, les élèves.

L'intitulé « étude exploratoire » de ce mémoire reflète le vaste potentiel du sujet, raison pour laquelle il nous a fallu choisir une direction à prendre et un objectif à atteindre. En premier lieu, nous avons décidé de nous baser sur la situation de la France puisque, contrairement à la Belgique, l'algorithmique fait partie intégrante des programmes du cours de mathématiques, entre autres. Ensuite, nous avons défini notre objectif principal, l'analyse à priori d'une séquence de cours portant sur l'algorithmique et la programmation. Pour atteindre cet objectif, il nous fallait donc nous intéresser de plus près à l'enseignement de l'algorithmique et la programmation en France, mais aussi définir un certain nombre de cadres théoriques afin de soutenir notre analyse.

Dans le chapitre 1 nous définissons un premier cadre théorique. Nous y présentons les notions principales liées à l'algorithmique et à ses objets d'étude, les algorithmes. Ce chapitre constitue la base de notre travail de recherche et d'analyse puisqu'il contient l'ensemble des définitions, des caractéristiques et des aspects relatifs à l'algorithmique que nous y retrouverons, ainsi que les liens entre l'algorithmique et la programmation.

Le chapitre 2 concerne le cas spécifique de la France. Nous y détaillons les événements qui ont mené à l'introduction de l'algorithmique dans les programmes ainsi que les motivations de ces changements. Nous effectuons un état des lieux des programmes du lycée et identifions, pour chacun d'entre eux, la façon dont l'algorithmique est intégrée ainsi que l'image qui en est donnée. Nous analysons aussi un document ressource à destination des enseignants et en dégageons un bilan.

Les chapitres 3 et 4 constituent nos deux autres cadres de référence. Le premier porte sur la didactique des mathématiques et nous y abordons un ensemble de notions liées à la théorie anthropologique du didactique de Chevallard et à la théorie des situations didactiques de Brousseau. Le second est quant à lui spécifique à l'utilisation des TICE dans l'enseignement. Ce cadre didactique des TICE aborde des théories telles que la genèse instrumentale de Rabardel, la transposition informatique de Balacheff et la pseudo-transparence d'Artigue. Il se termine par une brève application au cas de l'algorithmique et de la programmation.

C'est au sein du chapitre 5 que nous effectuons notre analyse. Tout d'abord nous présentons les différentes hypothèses qui ont été faites. Ensuite, nous donnons de plus amples informations sur le contexte de la séquence, nous identifions notamment les objectifs d'apprentissage ou encore les prérequis considérés, et nous analysons cette dernière au regard de nos différents cadres théoriques. Nous terminons par une conclusion générale du travail qui a été effectué, proposant notamment les limites de ce dernier ainsi que les perspectives qui en découlent.

Chapitre 1

Notions d'algorithmique

Ce premier chapitre a pour but de fixer les notions, les termes et le vocabulaire spécifique à l'algorithmique et à la programmation que nous rencontrerons tout au long de ce travail. Nous y définissons l'algorithmique, ainsi que les objets qu'elle concerne, les algorithmes. Nous prendrons le temps d'en extraire les caractéristiques et de parcourir les cinq aspects qui en ressortent. Ensuite nous discuterons de l'algorithme-objet et l'algorithme-outil ainsi que du lien entre algorithmes et programmation. Pour ce faire, nous nous basons sur la thèse de Simon Modeste publiée en 2012, « Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ? » [Modeste, 2012] et sur celle de Nathalie Briant un an plus tard, « Étude didactique de la reprise de l'algèbre par l'introduction de l'algorithmique au niveau de la classe de seconde du lycée français » [Briant, 2013]. Dans ce chapitre et les suivants, les citations provenant des différentes sources sont présentées sous la forme de paragraphes en retrait ou, lorsqu'elles se trouvent au sein même du texte, entre guillemets.

1.1 Définitions et caractéristiques

La première question à laquelle il nous faut répondre avant d'aller plus loin est : qu'est-ce que l'algorithmique ? Selon Simon Modeste [Modeste, 2012], il s'agit de la science qui s'évertue à étudier les algorithmes, tant au niveau de leur création et leur description que leur analyse. Mais dès lors, qu'est-ce qu'un algorithme ? Au fil du temps, une multitude de définitions de ce concept ont émergé et évolué, certaines relevant plus de l'énumération de ses propriétés que d'une définition au sens usuel, d'autres reléguant l'algorithme à sa simple fonction de résolution de problèmes destiné à être exécuté par un ordinateur. En recensant et analysant plusieurs de ces définitions, Modeste en a dégagé sa propre version censée prendre en compte tous les éléments importants qu'il avait pu identifier jusqu'à présent. C'est sur cette définition que nous allons nous baser pour la suite.

> Un **algorithme** est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème pour toute instance de cette famille. [Modeste, 2012]

La compréhension de la notion d'algorithme pouvant difficilement se reposer uniquement sur une définition, aussi complète soit-elle, nous allons en expliciter les différents éléments.

Instance et famille d'instances

En substance, Modeste décrit l'algorithme comme une « procédure de résolution de problèmes, s'appliquant à une famille d'instances du problème » et proposant une réponse à ce problème pour chaque « instance de cette famille ». Ce résumé de la définition demande déjà à lui seul quelques éclaircissements. En effet, non seulement la fonction première d'un algorithme y est précisée – la résolution de problème – mais en plus il est question de la portée de cette dernière, qui doit être valable pour toute instance d'une famille donnée. Définissons ce que l'auteur entend par **famille d'instances** et **instance**. Dans le glossaire disponible en annexe de sa thèse, Modeste propose la définition suivante du terme instance :

> Un problème p étant l'expression d'une même question Q pour tout élément d'un ensemble I donné, on appelle instances les éléments de I. [Modeste, 2012]

En d'autres mots, une famille d'instances d'un problème correspond à un ensemble de configurations possibles de ce dernier, tandis qu'une instance n'est autre qu'un élément de cet ensemble. Puisqu'un algorithme doit être capable de résoudre un problème pour chaque instance de la famille d'instances qui lui est associée, on peut le qualifier de général. Dès lors, une procédure qui ne serait pas générale, autrement dit qui ne résoudrait un problème que pour une instance spécifique, ne peut être considérée comme un algorithme. Modeste appelle cette procédure particulière un **algorithme-instancié**. Dans son mémoire, Meurist [Meurist, 2014] prend l'exemple de la recette de cuisine pour illustrer la notion d'algorithme-instancié.

> C'est notamment le cas de la recette de cuisine qui est souvent citée comme exemple pour illustrer la notion d'algorithme, alors qu'elle n'est valable que pour un plat particulier. [Meurist, 2014]

Entrées et sorties

Les notions d'instances et de familles d'instances sont étroitement liées à celles d'**entrées** et de **sorties** d'un algorithme, bien que ces dernières n'apparaissent pas explicitement dans la définition. Les entrées d'un algorithme sont communiquées à l'algorithme en amont de son exécution et sont associées à l'instance considérée. Elles peuvent être interprétées comme les données initiales du problème. Les sorties d'un algorithme désignent les données finales, celles renvoyées une fois que la procédure de résolution a pris fin. Par conséquent, une procédure ne comprenant pas d'entrée et/ou de sortie n'est pas un algorithme.

Conditions sur les étapes

Le reste de la définition concerne les étapes constitutives d'un algorithme. Ces dernières doivent être « en un nombre fini, constructives, effectives, non-ambiguës et organisées ». Toutes ces conditions sont indispensables pour qu'un algorithme puisse être mis en oeuvre de façon effective par un opérateur, qu'il soit humain ou machine. De fait, pour être exécutées correctement, il est impératif que les étapes d'un algorithme soient compréhensibles pour l'exécutant. En d'autres mots, il faut qu'elles soient non-ambiguës et

composées d'actions que Modeste [Modeste, 2012] qualifie d'élémentaires pour l'opérateur. Dans cette optique, l'organisation des étapes d'un algorithme est aussi un point crucial puisque, premièrement, elle permet de déterminer à chaque instant quelle étape doit être effectuée et, deuxièmement, un algorithme « n'est plus le même ou ne fonctionne plus si on intervertit des actions élémentaires » [Briant, 2013]. Un algorithme respectant tous ces critères peut donc être réalisé par un individu dans un environnement papier-crayon en un temps fini.

1.2 Cinq aspects de l'algorithme

Différents points que nous venons d'aborder, ainsi que d'autres dont nous n'avons pas encore parlé, sont résumés par Simon Modeste dans sa thèse [Modeste, 2012]. Il y présente ce qui, pour lui, constitue les cinq aspects fondamentaux des algorithmes : l'aspect problème, effectivité, preuve, complexité et modèles théoriques. Nous allons passer ces différents aspects en revue brièvement.

Aspect problème

Ce premier aspect fait référence au but premier d'un algorithme qui est la résolution d'un problème pour une famille d'instances, ce que Modeste désigne aussi par le fait de répondre à une question précise posée pour cette famille. On retrouve aussi les notions d'entrées et de sorties dont nous avons discuté précédemment, présentées par Modeste comme respectivement « l'instance à traiter » et « la réponse à la question pour cette instance » [Modeste, 2012].

Aspect effectivité

L'aspect effectivité d'un algorithme reprend l'ensemble des conditions d'effectivité que nous avons déjà rencontrées. Autrement dit, cet aspect présente l'algorithme comme une procédure effective dans le sens où il s'agit d'une procédure systématique traitant, en un nombre fini d'étapes non-ambiguës, un nombre fini de données et pouvant être exécutée par un opérateur quelconque. Dans le cas où l'opérateur est un ordinateur, l'algorithme doit être présenté dans un langage de programmation, sous la forme d'un programme, mais nous y reviendrons plus tard.

Aspect preuve

Les différentes propriétés d'un algorithmes peuvent être vérifiées grâce à un processus dit de **preuve d'algorithme**. Ce dernier comporte deux preuves distinctes. D'un côté nous avons la correction, qui consiste à montrer que, peu importe l'entrée, l'algorithme produit une réponse valide. De l'autre nous avons la terminaison, qui se concentre sur le fait de prouver que, peu importe l'entrée, l'algorithme produit bel et bien une réponse en un nombre fini d'étapes, autrement dit qu'il se termine. Différents autres points composent cet aspect mais nous ne les explorerons pas ici.

Aspect complexité

Une autre notion très importante en algorithmique est celle de complexité. Chaque algorithme présente sa propre complexité et cette dernière est déterminée en fonction de la taille de l'instance, « de l'ampleur des données entrées dans l'algorithme » [Meurist, 2014]. Modeste indique que la complexité représente en réalité le nombre d'opérations élémentaires effectuées par l'algorithme et que ce nombre peut être calculé dans le pire des cas ou en moyenne. Pour une instance donnée, le calcul pour le pire des cas consiste à déterminer le maximum d'opérations effectuées par l'algorithme, tandis que le calcul en moyenne, comme son nom l'indique, consiste à déterminer le nombre moyen d'opérations effectuées. Dans le cas d'un algorithme sous la forme d'un programme, la complexité peut être mesurée en temps ou en espace de stockage, autrement dit en taille de mémoire. Le calcul de la complexité constitue une bonne base pour la comparaison d'algorithmes lorsqu'on souhaite déterminer celui qui est le plus efficace, mais aussi pour « classifier les problèmes selon la complexité des algorithmes permettant de les résoudre » [Modeste, 2012].

Aspect modèles théoriques

Le dernier aspect concerne les modèles théoriques sur lesquels il est possible de se baser dans une démarche de classification d'algorithmes et de problèmes. Il est basé sur des modèles tels que les machines de Turing ou les fonctions récursives de Kleene pouvant être considérés, dans une certaine mesure, comme des définitions différentes de ce qu'est un algorithme. Ces derniers ont notamment permis de mettre en évidence l'existence de problèmes ne pouvant pas être résolus de façon algorithmique et de classifier ceux qui le sont bel et bien. Nous ne développerons pas plus en avant ce dernier aspect.

1.3 Dualité outil-objet

Les cinq aspects énoncés ci-dessus peuvent être classifiés en termes d'outils et d'objets. Concrètement, cela signifie que certains de ces aspects confèrent à l'algorithme un statut d'objet et d'autres un statut d'outil. Les premiers s'ancrent dans l'algorithmique et s'intéressent à la façon dont l'algorithme fonctionne, ses descriptions, sa complexité ou encore son domaine de validité, tandis que les seconds ne le considèrent qu'en tant qu'outil, un moyen pour résoudre des problèmes. Respectivement, les aspects de preuve, de complexité et de modèles théoriques se situent du côté de l'objet et ceux de problème et d'effectivité du côté de l'outil. Le tableau de la figure 1.1 résume ces positions.

Outil	Objet
Problème Effectivité	Preuve Complexité Modèles théoriques

Figure 1.1 – Dualité outil-objet des aspects de l'algorithme par Simon Modeste

Dans ses travaux, Modeste utilise ces cinq aspects de l'algorithme comme une sorte de grille d'analyse, permettant de mettre en évidence la façon dont les discours, les ressources ou les programmes d'enseignement perçoivent et présentent les algorithmes. En effet, « en relevant les aspects présents dans un discours ou un document, on peut étudier quelle vision de l'algorithme est en jeu et si les aspects outils et objets sont présents » [Modeste, 2012]. Nous aurons nous aussi l'occasion de nous servir de cette grille plus en avant dans ce travail.

1.4 Algorithme et programmation

Un algorithme peut être spécifié de différentes façons et revêtir de multiples formes (diagramme, langage naturel, langage mathématique, etc), mais cela dépend « de l'opérateur à qui cet algorithme est destiné et de la technologie qui permet d'exécuter l'algorithme » [Modeste, 2012]. Prenons l'exemple d'un algorithme destiné à être implémenté sur un ordinateur. Pour que ce dernier soit exécutable par la machine, il est nécessaire qu'il soit décrit dans un langage que cette dernière est en mesure de comprendre, c'est-à-dire dans un langage de programmation. À l'instar d'un langage naturel tel que le français, un langage de programmation possède une grammaire et une syntaxe qui lui est propre. Dans la majorité des langages de programmation, il existe des instructions spécifiques permettant « de contrôler l'enchaînement des opérations » [Modeste, 2012] qui sont exécutées par l'ordinateur. Briant [Briant, 2013] regroupe ces instructions en quatre catégories distinctes : l'affectation de variables, la lecture et l'écriture, les tests (structures conditionnelles) et les boucles (structures itératives). Il s'agit là de notions centrales de la programmation.

L'affectation

L'**affectation de variables** est associée à la notion de **variable informatique**. Une variable informatique est « un modèle de la mémoire d'une machine » [Modeste, 2012], autrement dit elle désigne un emplacement de la mémoire de l'ordinateur. Une valeur est associée à cette variable par le biais d'une opération d'affectation, mais cette dernière peut être modifiée à tout instant via le même processus.

La lecture et l'écriture

La **lecture** et l'**écriture** sont englobées dans le deuxième type d'instructions exécutables par un ordinateur car elles relèvent de la communication entre l'utilisateur et la machine. En effet, la lecture permet à l'utilisateur de communiquer au programme les données à utiliser, tandis que l'écriture communique des données à l'utilisateur. Dans le premier cas, les données doivent être encodées au clavier, dans le second elles sont affichées à l'écran.

Les tests

Ce que Briant appelle les **tests** fait en réalité référence aux structures conditionnelles de la forme "IF (condition A) THEN (instructions B) ELSE (instructions C)", pouvant s'interpréter en français comme "si la condition A est vérifiée, alors on effectue la séquence d'instructions B, sinon on effectue la séquence d'instructions C". Notons que la forme d'une structure conditionnelle n'est pas fixée à celle que nous venons d'énoncer. Par exemple, plusieurs conditions peuvent être testées les unes après les autres, ou encore la partie "sinon" peut être absente et il n'y a qu'une seule série d'instructions. La ou les condition(s) testée(s) dans ce type de structure sont particulières dans la mesure où il s'agit de booléens. Un booléen étant une expression ne pouvant prendre comme valeur que "vrai" ou "faux", dans laquelle apparait au minimum un opérateur de comparaison tel que "égal à", "différent de", etc.

Les boucles

Les boucles quant à elles sont des structures itératives qui, comme leur nom l'indique, permettent de répéter des opérations. Il existe deux types de boucles : les boucles "FOR" ("pour") et les boucles "WHILE" ("tant que"). Les premières répètent la séquence d'opérations un nombre précis de fois, tandis que les secondes la répètent uniquement tant qu'une condition donnée est vérifiée, c'est-à-dire qu'elle reste vraie. Ainsi, pour les boucles WHILE, le nombre de répétition qui vont être effectuées n'est pas connu à l'avance.

Bien entendu, il existe une multitude de langage de programmation différents, avec leur structure, leurs conventions d'écriture, leurs possibilités et leurs limites. Les programmes informatiques que ces derniers permettent d'écrire sont des entités exécutables constituées de « combinaison de ces quatre éléments de base [affectation, lecture et écriture, tests et boucles], allant de quelques-uns à plusieurs milliers » [Briant, 2013]. Cependant, un programme ne représente pas forcément un algorithme et un algorithme n'est pas nécessairement spécifié sous la forme d'un programme, il s'agit là de deux objets différents. Par ailleurs, remarquons que Simon Modeste ne fait aucun lien entre algorithmique et programmation dans sa définition à la page 5. Il s'agit là d'un choix délibéré de sa part, appuyant l'existence des algorithmes en dehors de tout environnement informatisé. Cela fait écho au fait que l'algorithmique peut tout à fait être envisagée sans avoir recours à une quelconque machine puisque, nous le savons à présent, un algorithme se doit d'être exécutable par un opérateur en un temps fini. La citation de Briant ci-dessous appuie cet argument.

> Un algorithme ne se définit pas forcément avec un langage de programmation et une simple feuille de papier et un crayon suffisent à en établir un. Un algorithme n'est pas synonyme de « programme informatique », il est indépendant du langage de programmation informatique utilisé pour l'écrire et l'exécuter. [Briant, 2013]

Enfin, Modeste nous dit que « un algorithme peut aussi être exprimé dans un langage bien moins formalisé, tel que la langue française ou le langage mathématique » [Modeste, 2012]. À cela, il ajoute l'existence d'un autre langage, souvent utilisé pour décrire les algorithmes,

appelé le « pseudo-code ». Ce langage, à mi-chemin entre langage naturel, mathématique et de programmation, a pour caractéristique d'être relativement proche du langage de programmation tout en étant « libéré de certaines contraintes et manipulant directement les objets mathématiques » [Modeste, 2012]. Par nature, le pseudo-code est « purement conventionnel » [Briant, 2013], c'est-à-dire que l'écriture d'algorithmes dans ce langage varie en fonction des individus.

En résumé, dans ce chapitre nous avons défini un premier cadre théorique, celui de l'algorithmique. Nous y avons présenté l'ensemble des notions liées à l'algorithmique et la programmation que nous allons rencontrer dans la suite de ce document. Globalement, la définition que nous avons choisie pour l'algorithme présente ce dernier comme une procédure de résolution de problème, pouvant être appliquée à une famille d'instances de ce dernier et proposant, en un temps fini, une solution à toutes les instances de ce problème. Mais en réalité le concept d'algorithme englobe plus que cela. En effet, nous avons vus qu'il est possible d'en dégager cinq aspects principaux – l'aspect problème, effectivité, preuve, complexité et modèles théoriques – mais aussi qu'un algorithme peut être considéré en tant qu'outil ou en tant qu'objet. Toutes ces notions, que nous maitrisons à présent, constituent notre cadre de référence et vont servir à identifier quelle vision de l'algorithmique est véhiculée dans les programmes de cours français qui, comme nous le verrons dans le chapitre suivant, ont déjà intégré l'algorithmique et la programmation dans leur enseignement depuis plusieurs années. Cet état des lieux de la situation en France nous mènera ensuite à définir d'autres cadres théoriques, spécifiques cette fois à la didactique et aux technologies dans l'enseignement et qui, en complément de celui que nous venons de poser, nous permettrons à terme d'effectuer l'analyse d'une séquence de cours.

Chapitre 2

Situation en France

Dans ce chapitre nous allons nous intéresser de plus près à ce qu'il se passe en France au niveau de l'enseignement de l'algorithmique. Nous allons détailler les réflexions et événements qui ont mené à l'introduction d'activités algorithmiques dans les cours proposés au lycée mais aussi nous attarder sur l'analyse des programmes d'enseignement des mathématiques dans les différentes filières. Pour permettre un meilleure compréhension de ces dits programmes, nous nous attarderons aussi sur l'enseignement de l'algorithmique dispensé au collège. Nous terminerons ensuite par l'étude d'un document ressource destiné aux enseignants du lycée.

2.1 Introduction de l'algorithmique en France

L'apparition de l'algorithmique dans les programmes français est due à un rapport émis par la "Commission de réflexion sur l'enseignement des mathématiques" (la CREM), en décembre 2000 [Kahane, 2000]. Ce rapport faisait suite à la création de cette commission (dite "Kahane", ce qui provient du nom de son président, Jean-Pierre Kahane) en avril 1999, à la demande d'associations de professeurs de mathématiques et de chercheurs. L'objectif de la commission Kahane était alors de mener une réflexion à long terme sur l'enseignement des mathématiques, de l'école élémentaire à l'université. Ce travail devait être effectué en amont de celui de l'élaboration des programmes de mathématiques de l'enseignement secondaire. Dans la suite, l'ensemble des citations proviennent du rapport [Kahane, 2000] et sont présentées avec une indentation plus importante (ou entre guillemets lorsqu'il s'agit de phrases plus courtes).

Le rapport "Informatique et enseignement des mathématiques" à l'origine du changement dans les programmes français se penchait sur trois questions importantes. Cependant nous nous attarderons uniquement sur les deux premières, énoncées dans le document comme :

1. Pourquoi introduire une part d'informatique dans l'enseignement des sciences mathématiques et dans la formation des maîtres ?
2. Comment faire évoluer les programmes pour accompagner cette évolution ?

Les réponses à ces questions ont été développées par la commission Kahane sans pour autant trop entrer dans les détails des interactions possibles entre l'activité mathématique et informatique, ni même parler de l'utilisation de logiciels.

Nous nous intéressons plus particulièrement aux conclusions de la commission concernant la première question. Globalement, il est ressorti de leur réflexion qu'il était nécessaire d'intégrer une part d'informatique dans l'enseignement des mathématiques au lycée et, pour ce faire, ils proposèrent alors des objectifs à court et moyen terme.

> À court terme : (...) Les programmes passés et actuels comportent des instructions précises qui sont peu ou pas appliquées, qui restent à l'état de recommandations sans articulation réelle avec le reste du programme. On peut donc commencer par donner vie dès maintenant à ces instructions en signalant les utilisations possibles et en les intégrant dans le texte proprement dit et les documents d'accompagnement des programmes.
>
> À moyen terme : Faire évoluer progressivement les contenus pour intégrer de nouveaux objets et notions (...) : algorithmique et programmation, mathématiques de l'informatique.

L'objectif global est donc de modifier les programmes et documents annexes déjà présents, de sorte à inciter à la mise en pratique de certaines consignes qui n'étaient pas toujours intégrées aux cours. En quelque sorte, le but était de forcer l'utilisation de ces instructions dans les programmes. D'un autre côté, nous comprenons que même si l'idée principale était effectivement d'intégrer l'algorithmique et l'informatique conjointement aux mathématiques, il était aussi question de mettre en place une évolution progressive des anciens contenus vers les nouveaux.

2.1.1 Motivations de la commission

Les chercheurs et enseignants composant la commission motivèrent leurs recommandations en mettant en évidence les liens particuliers entretenus par les mathématiques avec l'informatique, ainsi que l'ensemble des nouvelles possibilités qu'amène l'introduction de ces nouvelles notions.

> Dans son communiqué n°5, notre commission écrivait :« L'informatique donne des motivations et un champ nouveau à l'enseignement des mathématiques. Elle amène à revisiter des notions anciennes, à introduire de nouveaux points de vue, à donner des éléments nouveaux aux professeurs et aux élèves. »

Dans la première partie du rapport, les auteurs discutent de l'esprit algorithmique pour lequel il est aisé de faire un parallèle avec le fonctionnement de notre propre esprit. En effet, chaque jour nous ne manquons pas de décomposer des tâches complexes en une succession de tâches plus petites et plus simples. Non seulement nous effectuons cette décomposition, mais nous savons estimer la durée d'un tel processus ainsi que vérifier si l'ensemble s'est bien déroulé et nous donne le résultat escompté. De plus, nous sommes aussi capables d'identifier l'utilité d'effectuer une action de façon répétée ainsi que de repérer les tâches qui sont déjà terminées. Tout cela s'apparente énormément à la démarche

sous-jacente à l'écriture d'un programme et à son exécution. Tous ces efforts de réflexion et de construction de la pensée peuvent donc être mis en œuvre de la même façon pour l'écriture d'un programme, sans oublier de s'inquiéter que le tout s'effectue — et se termine — dans un temps raisonnable.

Un autre point important discuté dans ce rapport est l'utilisation de l'informatique et des ordinateurs dans le calcul et le traitement des données. Par exemple, que ce soit en économie, en médecine, en météorologie ou encore en géologie, il est très utile de pouvoir disposer d'un outil assez puissant pour, à l'aide d'un modèle mathématique, simuler le comportement d'un système complexe. Bien évidemment, les mathématiques n'échappent pas à cette règle. En effet, ce serait oublier toute une partie des mathématiques que de se limiter aux résultats qu'il est possible de prouver ou représenter en papier-crayon. Pour illustrer cet état de fait, le rapport cite l'étude théorique des équations aux dérivées partielles qui nécessite de façon systématique l'aide de la simulation numérique.

Outre l'ordinateur, nous pouvons aussi citer un second outil qui a énormément évolué ces dernières décennies : la calculatrice. Elle figure sur toutes les listes de matériel scolaire et permet à elle seule d'effectuer des opérations de plus en plus complexes. En effet, il est maintenant possible d'y tracer des graphes, de construire des figures géométriques planes, de factoriser, de dériver, d'intégrer, de déterminer des développements de Taylor et même de simuler facilement le comportement de systèmes dynamiques relativement simples, comme précisé dans le rapport [Kahane, 2000]. Il s'agit là d'un deuxième outil ayant subi des modifications assez importantes pour que l'enseignement les prenne en considération.

Revenons maintenant sur l'intérêt d'enseigner ces concepts dans le cadre spécifique du cours de mathématiques. La réponse de la commission à cette question contient trois sous-points. Premièrement, la commissions met en évidence les liens qu'ils font naitre entre la logique, l'algorithmique et l'informatique, permettant ainsi de mettre ces différents domaines en action de façon simultanée. Deuxièmement, comme nous l'avons déjà explicité précédemment, ils permettent une meilleure compréhension des outils tels que les ordinateurs et les calculatrices, qui sont utilisés très majoritairement comme aide à l'enseignement des mathématiques. Troisièmement, ils permettent aussi d'expliquer et de développer certains concepts informatiques qui ont été empruntés aux mathématiques avant d'être remaniés, modifiés et codifiés pour coller au domaine dans lequel ils sont utilisés (la récurrence, la récursivité, la différence entre le nom d'une variable et sa valeur, ...).

En résumé, l'argument principal en faveur de l'introduction d'une part d'informatique dans l'enseignement des sciences mathématiques est que, de nos jours, « l'ordinateur est devenu un outil indispensable sur le bureau d'un mathématicien » [Kahane, 2000] et a si bien bouleversé notre vie — et la science de façon générale — qu'on ne peut décemment pas l'ignorer dans l'enseignement.

Pour finir, intéressons-nous rapidement aux logiciels qui sont proposés aux enseignants comme support ou comme outil pour leurs cours. La commission Kahane relève dans son rapport un point important les concernant.

> Il nous semble illusoire de penser que la convivialité des logiciels mathématiques rendra un jour inutile l'apprentissage des notions de base d'algorithmique et programmation. Pour que l'élève et l'enseignant aient une attitude active et imaginative, ils ne peuvent s'en remettre pieds et poings liés au fabricant de la machine ou à l'éditeur de logiciels.

En effet un tel comportement reviendrait à utiliser ces logiciels de la même façon que n'importe quelle personne a recours à une calculatrice : sans comprendre ce qui se cache derrière. Or l'un des objectifs majeurs de la réforme proposée par la commission est de ne pas laisser ce type d'outils incompris.

2.1.2 État des lieux avant la réforme

À présent que nous avons exposé les différentes motivations de la commission concernant l'introduction de l'informatique et de l'algorithmique dans l'enseignement, nous pouvons nous pencher sur l'évolution des programmes qui en a découlé. Cela concerne essentiellement la modification des contenus déjà présents et l'ajout de concepts d'algorithmique et de programmation. Nous allons discuter dans la suite de différents points de matière présents dans les programmes avant la réforme, en mettant en évidence les questions relatives à ces notions qui relèvent des domaines de l'informatique et de l'algorithmique.

Pour commencer, discutons de la manipulation et des calculs avec des nombres. Dans le rapport de la commission il est expliqué que, durant cette manipulation, des approximations peuvent apparaitre. Dans ces cas-là, des questions s'imposent naturellement : comment faut-il exprimer ces approximations ? Sont-elles fiables ? Comment le savoir ? Il s'agit là de savoir observer de manière critique les résultats obtenus, ce qui constitue une part importante de la réflexion en informatique et en algorithmique. Lorsqu'il est admis que nous travaillons avec des approximations, il est crucial de se demander quel est le format des données que nous utilisons, ainsi que celui des résultats que nous obtenons. En effet, cela a un impact sur l'erreur commise, qu'il faut alors estimer elle aussi, et nécessite de maitriser les ordres de grandeur. Une autre différence entre le calcul exact et le calcul avec approximations se situe dans l'interaction entre le choix de la structure des données et le choix de l'algorithme de calcul utilisé. Par exemple, calculer $(a+b)-c$ ne donne pas forcément le même résultat que $a+(b-c)$.

Supposons à présent que nous choisissons d'effectuer uniquement du calcul exact, dans ce cas les questions que nous devons nous poser sont les suivantes : avec quel type de nombre travaillons-nous ? Et dès lors, sous quelle forme devons-nous les représenter ? Prenons l'exemple de $\sqrt{2}$: va-t-on le choisir comme la racine positive de l'équation $x^2-2=0$ ou le point fixe de la fonction $x \to 1+\dfrac{1}{1+x}$? Sachant que se fixer sur l'un ou l'autre implique notamment d'utiliser différents algorithmes de calcul d'une approximation.

En ce qui concerne la manipulation d'objets par des logiciels de calcul, certains algorithmes utilisés font partie des fondamentaux du calcul numérique (pensons notamment à la méthode de Newton). Ils permettraient alors de se donner l'occasion d'observer des

systèmes dynamiques en temps discret, de se pencher sur les questions de rapidité de convergence ainsi que de « découvrir de nouveaux aspects des objets étudiés (par exemple, la méthode d'Euler éclaire le théorème d'existence et d'unicité des solutions d'une équation différentielle du premier ordre) » [Kahane, 2000].

Intéressons-nous finalement à la génération de nombres aléatoires. En temps normal lorsque nous souhaitons obtenir un nombre de façon aléatoire (ou une suite de nombres) nous utilisons des fonctions pré-installées sur nos logiciels ou une touche en particulier sur notre calculatrice. Cela revient parfois à n'utiliser qu'une simple table de nombres préétablie. L'intérêt alors d'intégrer le codage de générateurs de nombres aléatoires serait de permettre de ne pas se limiter à ce type de table. De plus, cela rendrait alors possible l'introduction de structures particulières telles que les listes, les arbres ou les graphes qui sont utiles dans ce genre de problème.

Dès lors, le rapport définit quels sont les concepts qu'il faut nécessairement ajouter aux programmes. Il s'agit de ceux qui vont permettre d'aborder la matière d'un point de vue informatique et algorithmique, comme présenté dans la section précédente. La commission indique que, majoritairement, ce sont les concepts de base de la programmation et l'algorithmique qu'il faut introduire. Autrement dit, nous parlons des manipulations de données (lecture, affectation de variables, écriture de données) et des structures de contrôle telles que les boucles et les branchements. Ensuite viennent, dans une moindre mesure, les instructions plus complexes (telles que la récursivité) ainsi que la structure des données et la complexité des algorithmes.

2.2 Analyse des programmes

2.2.1 Collège

Avant même de commencer leurs années de lycée, les élèves français se retrouvent déjà confrontés à des problèmes et des activités de type algorithmique. En effet, durant les trois années précédentes, au collège, la programmation et l'algorithmique font partie intégrante du programme soumis par la Direction générale de l'enseignement.

Par conséquent, nous allons explorer et analyser le programme de cours disponible pour le collège. Plus précisément, nous allons nous concentrer sur le cycle 4 (voir figure 2.1), qui représente l'équivalent des première, deuxième et troisième années de l'enseignement secondaire en Belgique. Les objectifs et l'ensemble des aspects relatifs à l'algorithmique et la programmation en seront dégagés, ce qui, avec l'analyse des programmes du lycée d'autre part, nous permettra d'avoir une vue d'ensemble sur toutes les notions d'algorithmique enseignées aux élèves durant leur scolarité en France. Toutes les citations présentes dans cette section proviennent donc d'une seule et même source : le programme des cours du cycle 4 [MENJ, 2018], en vigueur depuis la rentrée 2018-2019, proposé par le Ministère de l'Éducation Nationale et de la Jeunesse.

Le document présentant le programme est divisé en trois parties (appelées des volets) : « les spécificités du cycle des approfondissements », « contributions essentielles des diffé-

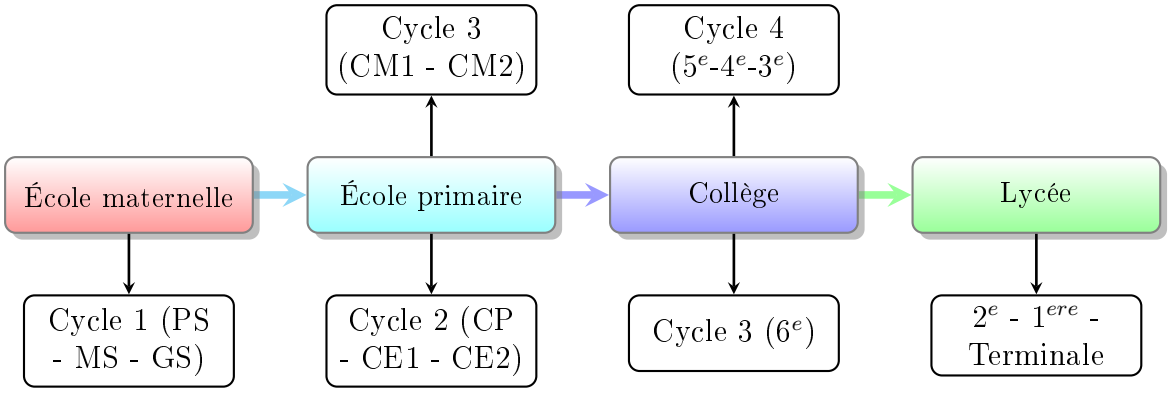


FIGURE 2.1 – Diagramme du système scolaire français de la petite section (école maternelle) à la terminale (lycée)

rents enseignements au socle commun » et « les enseignements ». Dans ce qui suit, nous allons développer des éléments provenant de chacun de ces volets.

Les spécificités du cycle des approfondissements

Ce premier volet présente les caractéristiques spécifiques du dernier cycle et, en surface, le programme qui lui est associé. Parmi les objectifs présentés pour ces trois dernières années du collège, nous pouvons remarquer que certains sont prometteurs, dans le sens où ils sont propices à des activités algorithmiques (nous verrons plus tard lesquelles). En effet, durant cette période de trois ans,

> Il s'agit de former les élèves à être capables de dépasser le cas individuel, de savoir disposer d'outils efficaces de modélisation valables pour de multiples situations et d'en comprendre les limites.

Il est aussi indiqué que « la créativité des élèves (...) se déploie au cycle 4 à travers une grande diversité de supports (notamment technologiques et numériques) ». Les supports qui sont mentionnés dans cette phrase ne sont pas plus développés, néanmoins nous aurons l'occasion par la suite de confirmer que l'algorithmique et la programmation en font partie.

Contributions essentielles des différents enseignements au socle commun

Le deuxième volet présente, comme son nom l'indique, les apports essentiels de chaque discipline à « l'acquisition de chacun des cinq domaines du socle commun de connaissances, de compétences et de culture [1] ». Parmi ces cinq domaines, nous n'en aborderons que trois, car les deux autres n'impliquent pas (de façon directe ou indirecte) d'algorithmique ou de programmation.

Le premier domaine concerne les langages et est intitulé « Les langages pour penser et communiquer ». Une des compétences attendues dans ce domaine est de comprendre

1. Le socle commun de connaissances, de compétences et de culture correspond à ce que tout élève français doit savoir et maitriser à la fin de la période d'instruction obligatoire.

et de s'exprimer dans différents langages, notamment les langages mathématique, scientifique et informatique. Dans cette optique, il est aussi indiqué que « les mathématiques, les sciences et la technologie (...) aident à passer d'une forme de langage courant à un langage scientifique ou technique et inversement ». Puisque l'algorithmique et la programmation demandent d'effectuer de telles traductions et transpositions d'un langage à un autre, il n'est donc pas étonnant de retrouver ces notions plus loin dans le programme (puisque il n'en a pas encore été fait mention jusque là).

Ensuite, le second des cinq domaines (« Les méthodes et outils pour apprendre »), nous apprend l'existence d'un enseignement de l'informatique durant le cycle 4. De plus, cet enseignement est donné de façon simultanée dans deux cours différents. En effet, ce dernier est dispensé à la fois au cours de mathématiques et au cours de technologie. Il est par ailleurs présenté comme un outil à la compréhension de la matière mais aussi comme permettant « d'approfondir l'usage des outils numériques et d'apprendre par essai erreur ».

Finalement, le dernier domaine qui nous intéresse dans cette partie est le quatrième : « Les systèmes naturels et les systèmes techniques ». C'est dans ce passage du programme que l'algorithmique est mentionnée pour la première fois. En effet, il est dit que

> Les sciences aident à se représenter, à modéliser et appréhender la complexité du monde à l'aide des registres numérique, géométrique, graphique, statistique, symbolique du langage mathématique. (...) Elles contribuent à former le raisonnement logique par le calcul numérique ou littéral, la géométrie et l'algorithmique.

Les programmes

Maintenant que nous avons passé en revue les deux premiers volets du document, nous allons pouvoir nous pencher sur les programmes des différents cours dispensés au cycle 4 du collège. Comme précisé dans le deuxième domaine, l'enseignement de l'informatique se fait à travers les cours de technologie et de mathématiques, nous allons donc logiquement nous intéresser uniquement au programme de chacun de ces deux cours.

Le cours de technologie

Avant de présenter le programme, une section spécifique est dédiée à la présentation du cours. Il y est d'ailleurs précisé tout d'abord que le cours de technologie est le prolongement de l'éducation scientifique et technologique des années et des cycles précédents. De plus, il est indiqué que le fil conducteur du cours de technologie est de confronter les élèves à des activités « d'investigation, de conception, de modélisation, de réalisation et aux démarches favorisant leur implication dans des projets individuels, collectifs et collaboratifs ». C'est lors de ces activités et projets que les élèves seront amenés à appréhender, comprendre et utiliser les notions introduites dans le cadre de l'enseignement de l'informatique dispensé dans ce cours. Le document apporte ensuite une motivation à l'ajout de l'informatique dans la matière du cours de technologie. En effet, il est dit que

> Il (l'enseignement de l'informatique) permet d'acquérir des méthodes qui construisent la pensée algorithmique et développe des compétences dans la représentation

> de l'information et de son traitement, la résolution de problèmes, le contrôle des résultats.

Ce qui signifie que l'un des objets de ce cours est de développer la pensée algorithmique et les compétences connexes liées à son application sur des problèmes concrets. Cela contribue à créer des bases solides pour la compréhension de l'ensemble des notions d'algorithmique qui seront abordées plus tard, au lycée.

Juste après la présentation du cours, ce sont les compétences travaillées durant celui-ci qui sont détaillées. Seule l'une d'entre elles implique de l'algorithmique, la quatrième : « pratiquer des langages ». L'aptitude à travailler est alors décrite comme l'application des principes élémentaires de l'algorithmique et du codage dans le but de résoudre un problème simple.

Ensuite, le programme est détaillé pour chacun des objectifs du cours. Nous allons donc dans la suite mettre en évidence les objectifs pour lesquels les connaissances et les compétences associées font référence à la programmation et à l'algorithmique. Le premier objectif est énoncé comme « imaginer des solutions en réponse aux besoins, matérialiser une idée en intégrant une dimension design ». Dans la case des compétences associées, il est précisé que les collégiens doivent être aptes à concevoir de telles solutions mais aussi à les représenter, notamment, sous forme d'algorithmes. Puisque ce programme concerne le collège, nous pouvons supposer que la forme algorithmique attendue se fait probablement dans un premier temps sur papier, en langage naturel, puis éventuellement dans un langage de programmation simple.

Deuxièmement, il est demandé des élèves qu'ils soient capables d'exprimer leur pensée à l'aide d'outils de description adaptés tels que des croquis à main levée, des schémas, des cartes heuristiques et enfin des notions d'algorithme. Il n'est cependant pas précisé de quelles notions il s'agit en termes d'algorithmique. De la même façon que pour le paragraphe précédent, nous ne savons pas sous quelle forme l'algorithme doit apparaitre.

Enfin, pour la partie du cours entièrement dédiée à l'informatique et la programmation, il est attendu de la part des collégiens qu'ils soient capables d'écrire, de mettre au point et d'exécuter un programme. Les connaissances et compétences associées à cet objectif du cycle 4 sont la maitrise de notions d'algorithmiques et de programmation (sans précision supplémentaire) ainsi que le déclenchement d'une action par un événement, les séquences d'instructions, les boucles et enfin les instructions conditionnelles. Cette fois-ci nous allons aussi regarder du côté des exemples d'activités proposées par le programme. Il est conseillé notamment d'amener l'élève à concevoir et programmer des applications informatiques qui seront utilisées sur des appareils nomades (tels que les smartphones et les tablettes) mais aussi à écrire des programmes à partir de consignes de fonctionnement (un cahier de charges) ou encore, à modifier un programme déjà existant. Il n'y a pas d'algorithme proposé en tant que tel, mais les élèves doivent en finalité être capables d'en créer par eux-mêmes à partir d'une situation, d'un contexte ou d'un problème donné.

À ce stade, nous ne savons donc pas encore quels types de programmes ou d'algorithmes sont censés être développés, cependant les repères de progressivité (selon l'année)

indiqués dans la suite du document permettent d'apporter une réponse à cette question. En cinquième, le cours doit se baser sur des programmes simples afin de discuter de leur traitement, leur mise au point et leur exécution lorsque le nombre de variables d'entrée et de sortie est relativement limité. Aussi, dès cette première année du cycle, les élèves apprennent à maitriser les boucles itératives. En quatrième, les programmes considérés doivent prendre en considération plus de variables afin d'augmenter le niveau de difficulté de façon progressive. Ce n'est qu'en dernière année que le comptage et les boucles imbriquées sont introduites, ainsi que la décomposition d'un problème important en plusieurs sous-problèmes.

Le cours de mathématiques

À présent nous pouvons nous pencher sur le cas du cours de mathématiques, le second cours qui dispense un enseignement de la programmation et de l'algorithmique. Nous pouvons d'ailleurs directement nous intéresser à la section du programme de mathématiques nommée « Algorithme et programmation ». Il s'agit de l'un des cinq thèmes abordés dans le cours, le seul qui implique explicitement de l'algorithmique.

En préambule de cette section, un paragraphe d'introduction explique que, durant ce quatrième cycle de scolarité, les élèves sont initiés à la programmation dans le cadre de projets nécessitant d'utiliser des programmes simples. Ils sont amenés à les créer de toutes pièces, à les modifier, voire à les compléter, ce qui leur permet de développer différentes méthodes de programmation tout en s'entrainant à adopter un raisonnement adapté aux tâches de type algorithmique. Il est aussi dit que l'un des autres avantages d'un tel apprentissage est qu'il permet de revisiter les notions de variables et de fonctions sous une forme différente, ce qui améliore leur compréhension de la part des élèves. Dans ce passage il est par ailleurs rappelé que le but final n'est pas la maitrise totale d'un langage de programmation en particulier, raison pour laquelle aucun langage n'est précisé. Certaines activités possibles sont ensuite listées, il s'agit uniquement de jeux (jeu de Pong, bataille navale, tic tac toe, ...) que les élèves seraient à même de comprendre facilement et de pouvoir coder par la suite.

À nouveau, le document nous indique les compétences attendues en fin de cycle. Pour la partie du cours de mathématiques qui nous intéresse, l'unique attente est que les élèves puissent « écrire, mettre au point (c'est-à-dire tester et corriger) et exécuter un programme simple ». Dans cette optique, les connaissances jugées comme nécessaires sont les suivantes : des notions d'algorithmique et de programmation, des notions de variables informatiques, le déclenchement d'une action par un événement et la maitrise des séquences d'instructions, des boucles et des instructions conditionnelles. Nous pouvons remarquer qu'il s'agit des mêmes attentes que pour le cours de technologie.

Bilan du cycle 4

Avec l'analyse de ces deux programmes du cycle 4 du collège, nous avons donc pu constater la présence de l'algorithmique et de la programmation ainsi que leur importance dans le cursus. Cependant, contrairement à ce que nous verrons dans la suite pour le lycée, ces notions ne sont pas forcément mélangées de façon homogène à la matière en-

seignée dans les autres parties des cours. En effet, dans le cas du cours de mathématiques, il s'agit d'un thème à part entière qui propose de développer les connaissances à travers des projets de conception de jeux aux règles simples, qui ne nécessiteront pas nécessairement d'utiliser des connaissances purement mathématiques. En général, il ne s'agit pas non plus d'implémenter des méthodes de résolution de calculs ou de problèmes particuliers en lien avec de la matière vue dans un des autres thèmes (« nombres et calculs », « organisation et gestion de données, fonctions », « grandeurs et mesures », « espace et géométrie »).

Pour le cours de Technologie par contre, les aptitudes attendues sont directement en lien avec le contenu du cours. L'approche est donc différente dans chacun des deux cours, bien que l'apprentissage se déroule de façon parallèle. Dans les pages qui suivent, nous verrons comment l'organisation de l'enseignement de l'algorithmique et de la programmation évolue avec le passage au lycée. Dans tous les cas, nous savons donc que les élèves disposent d'un certain bagage de connaissances en termes d'algorithmique, acquis durant leurs années de cinquième, quatrième et troisième au collège.

2.2.2 Lycée

C'est à partir de 2009, suite à la publication du rapport de la commission en 2000, que les programmes de mathématiques de la voie générale du lycée en France ont été modifiés en intégrant des parts d'algorithmique. Dans cette partie du chapitre nous allons donc nous concentrer sur ces nouveaux programmes et les analyser.

Le lycée est composé de trois années : la seconde (commune pour les voies générale et technologique), la première et la terminale (qui composent à elles deux le cycle terminal). C'est à partir du cycle terminal qu'on retrouve les différentes filières d'enseignement général : la série économique et sociale (ES), la série littéraire (L) et la série scientifique (S).

L'enseignement des mathématiques n'est pas donné de façon identique dans toutes les filières. Dans le diagramme représenté à la figure 2.2, "Commun" signifie qu'il s'agit d'un enseignement obligatoire pour tous les élèves de seconde, "Spécifique" que l'enseignement est obligatoire en terminale et "Spécialité" indique un enseignement optionnel en supplément de l'enseignement spécifique. De plus, lorsque plusieurs filières sont désignées par la même flèche, cela signifie que la matière abordée dans les cours est identique pour les différentes séries. Enfin, le cas particulier (indiqué par une flèche en pointillés) de l'enseignement de spécialité en terminale pour la filière L est que le programme du cours est le même que celui de l'enseignement spécifique de terminale ES.

Nous allons donc analyser les six documents correspondants aux programmes suivants :

- Programme de mathématiques de seconde - 23 juillet 2009,
- Programme de mathématiques de première ES et L - 30 septembre 2010,
- Programme de mathématiques de première S - 30 septembre 2010,
- Programme de mathématiques de terminale ES et L - 13 octobre 2011,

- Programme de mathématiques de terminale S - 13 octobre 2011,
- Programme de l'enseignement de spécialité terminale ISN - 01 aout 2017.

Le programme de terminale Informatique et Sciences du Numérique (ISN) est un cas à part que nous développerons plus tard. D'ailleurs, le document que nous allons analyser pour cette série est plus récent que les autres, il est de 2017, sa précédente version datant de 2011.

Pour l'ensemble de l'étude des programmes nous nous basons fortement sur la grille d'analyse et les conclusions fournies par Simon Modeste dans sa thèse intitulée "Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?" [Modeste, 2012].

2.2.3 Objectifs en matière d'algorithmique pour le lycée

Dans chacun de ces programmes, il y a un paragraphe dédié à la place de l'algorithmique au lycée ainsi qu'aux objectifs à atteindre durant ces trois années scolaires. Celui-ci relate notamment les différentes fois au cours desquelles les élèves ont eu l'occasion de rencontrer des problèmes de type algorithmique, déjà au collège par exemple.

Ces objectifs à atteindre sont de familiariser les élèves avec les principes fondamentaux des algorithmes et de leur organisation. Autrement dit, la gestion des entrées-sorties, l'affectation d'une valeur et la mise en forme d'un calcul. La formalisation des algorithmes quant-à-elle doit se faire dans un langage dit "naturel", qui puisse se traduire aisément en un langage propice à la programmation sur une calculatrice ou à l'aide d'un logiciel.

En finalité, les activités algorithmiques proposées dans les programmes ont comme effet d'entrainer les élèves à décrire des algorithmes en langage naturel et symbolique mais aussi à réaliser des algorithmes avec l'aide d'un tableur ou d'un petit programme, le tout

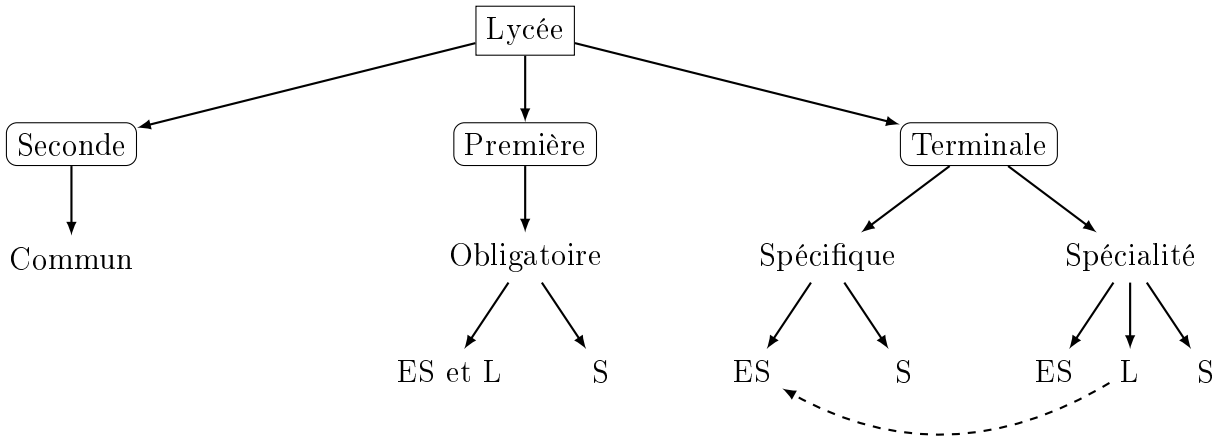


Figure 2.2 – Diagramme de l'enseignement des mathématiques au lycée pour les voies générale et technologique

sur une calculatrice ou en utilisant un logiciel adapté et, enfin, interpréter des algorithmes plus complexes. Ces exercices de description et d'écriture permettent alors d'apprendre aux élèves la rigueur et de les former à la vérification ainsi qu'au contrôle systématique des algorithmes qu'ils rencontrent ou mettent en place.

Il est aussi précisé que les problèmes posés doivent impérativement être en relation avec les autres parties du programme de mathématiques, qui sont les fonctions, la géométrie, les statistiques et probabilités ainsi que la logique. En effet, la grosse nouveauté dans ces programmes est que l'algorithmique apparait comme un thème transversal qu'il faut intégrer chaque année et dans tous les chapitres. Notons d'ailleurs qu'aucun logiciel ou langage n'est imposé par les auteurs.

2.2.4 Programme de seconde

Comme dit plus haut, le programme de mathématiques prévoit que de l'algorithmique soit introduite dans chacune des grandes parties du programme. En ce qui concerne le programme de seconde, il s'agit des fonctions, de la géométrie ainsi que des statistiques et probabilités. En se basant sur ce dit programme, nous pouvons citer ici différents exemples d'applications algorithmiques qui sont proposées, selon le point de matière à aborder. Le même travail sera effectué dans la suite pour toutes les autres années et filières.

Fonctions

Pour l'étude qualitative des fonctions, il est précisé qu'il serait intéressant de faire écrire aux élèves un algorithme permettant de tracer les courbes sans passer par un logiciel dédié. Il s'agit donc du processus de création et d'écriture d'un algorithme dans un environnement de programmation, comme souhaité par la commission Kahane, bien qu'il ne soit pas précisé si l'objectif visé est de tracer simplement la fonction à partir de son équation (calculer les valeurs points par points) ou de se pencher sur le cas plus spécifique des fonctions définies par morceaux et les conditions qui y sont associées. L'algorithmique est donc ici présentée en tant qu'outil, celui permettant d'afficher une courbe.

Dans un second temps, il est demandé que les lycéens soient capables « d'encadrer une racine d'une équation grâce à un algorithme de dichotomie » [MENJ, 2009a]. On remarque qu'il n'est pas précisé si la capacité attendue se situe au niveau de la conception d'un tel algorithme ou de la résolution de l'équation. Il est aussi à noter que rien n'est proposé pour mettre en avant l'algorithme dichotomique en tant que centre d'intérêt à part entière. Autrement dit, rien ne pousse le professeur à poser des questions sur cet algorithme (par exemple sur son efficacité, ses limites, son optimalité...). Cependant, il est vrai que de tels questionnements peuvent s'avérer plus intéressants à aborder en première ou en terminale plutôt qu'en seconde.

Géométrie

En géométrie, le programme de seconde consiste à étudier les configurations du plan. Le commentaire associé nous dit que

> Le cadre de la géométrie repérée offre la possibilité de traduire numériquement des propriétés géométriques et permet de résoudre certains problèmes par la mise en œuvre d'algorithmes simples. [MENJ, 2009a]

Il n'y a pas de précision supplémentaire sur le choix des algorithmes à présenter, il pourrait très bien s'agir d'algorithmes de vérification de propriétés géométriques ou de détermination de coordonnées de points particuliers (milieu, intersection de deux droites) afin de résoudre certains problèmes. Cependant que ce soit l'un ou l'autre, cela se rapporte à la traduction de formules mathématiques en programmes et nous retrouvons à nouveau une utilisation de l'algorithmique comme un outil au service des mathématiques.

Statistiques et probabilités

Dans cette dernière partie, deux applications sont proposées. La première concerne la réalisation d'une simulation, pour laquelle il serait possible d'introduire les instructions conditionnelles dans un algorithme. Il est clair que dans ce cas précis le but recherché est alors d'apprendre la programmation.

La seconde quant-à-elle est présentée dans le cas de la répétition d'expériences aléatoires. En effet, il est alors tout à fait possible de faire écrire des algorithmes. L'exemple donné est celui des marches aléatoires. Nous nous retrouvons à nouveau avec un objectif d'écriture d'algorithme qui est, cette fois-ci, motivé par le fait de répéter de façon conséquente une même expérience aléatoire.

2.2.5 Programmes de première

Séries ES-L

Pour le cours de mathématiques de la première en filières ES et L, les grandes parties du programmes sont l'algèbre et l'analyse, ainsi que les statistiques et probabilité. Ces deux sections sont présentées ensemble puisque, comme le montre le diagramme 2.2, les programmes sont communs.

Algèbre et analyse

Pour le second degré, le programme ne fait que préciser que "des activités algorithmiques sont réalisées dans ce cadre" [MENJ, 2010a], ce qui n'aide pas à identifier clairement le but de l'introduction de l'algorithmique pour ce point de matière. Nous pouvons néanmoins imaginer que l'une des activités possibles serait d'implémenter différentes formules.

Le prochain contenu pour lequel de la programmation et de l'algorithmique sont proposées est celui des suites. Parmi les capacités attendues dans cette partie, l'une d'entre elles est d'écrire et utiliser un algorithme ou un tableur afin de calculer un terme de rang donné d'une suite, ce qui permettrait d'étudier tout particulièrement des suites générées par une relation de récurrence. Il est aussi écrit dans le programme qu'il est possible de « traiter des problèmes de comparaison d'évolutions, de seuils et de taux moyens » [MENJ, 2010a]

à l'aide d'un tableur ou d'un algorithme. À nouveau, il s'agit d'implémenter des formules dans une démarche expérimentale d'analyse d'une suite ou de comparaison de deux suites.

Statistiques et probabilités

Dans cette deuxième et dernière partie du programme de mathématiques de ces deux filières en première, l'algorithmique apparait dans deux types de contenus. En probabilités, il est simplement dit qu'il est possible de simuler la loi binomiale avec un algorithme. L'algorithme est donc utilisé comme un outil permettant la répétition d'un certain nombre d'expériences aléatoires, comme nous l'avions déjà remarqué précédemment. Un peu plus loin, il est question de la détermination de l'intervalle de fluctuation à l'aide d'un tableur ou d'un algorithme. Dès lors, l'utilisation spécifique d'un algorithme à la place d'un tableur n'étant pas nécessaire, l'idée serait plutôt d'appliquer la formule de l'intervalle de fluctuation dans le tableur.

Série S

Le programme de mathématiques de la première en section S est identique à celui des sections ES et L pour la totalité de la partie analyse mais diffère légèrement dans la partie statistiques et probabilités. En effet, l'ajout qui a été fait en termes d'algorithmique est que "on peut simuler la loi géométrique tronquée avec un algorithme" [MENJ, 2010b]. La démarche est la même que pour la simulation d'une loi binomiale, présente elle aussi dans le programme de la filière scientifique.

2.2.6 Programmes de terminale

Séries ES et L

À nouveau, les programmes de terminale en mathématiques sont identiques pour les filières ES et L, nous les présentons donc de façon indistincte.

Analyse

L'unique contenu du programme, pour la partie analyse, pour lequel une activité algorithmique est proposée est celui qui porte sur les suites (géométriques en particulier) avec l'indication suivante en tant que capacité attendue :

> Étant donné une suite (q^n) avec $0 < q < 1$, mettre en œuvre un algorithme permettant de déterminer un seuil à partir duquel q^n est inférieur à un réel a positif donné. [MENJ, 2011a]

Ici nous constatons que l'objectif est de développer une capacité bien précise liée au traitement des suites. En termes d'algorithmique une telle attente devrait probablement se traduire par l'introduction des boucles "WHILE". De nouveau rien ne pousse à entrer dans les détails de l'algorithme, c'est-à-dire son efficacité, sa complexité, etc... Nous verrons par la suite que ce dernier point notamment fait l'objet d'un enseignement spécifique dans un autre cours du lycée.

Enseignement de spécialité

Enfin, dans la partie du programme réservée à l'enseignement de spécialité en mathématiques de la série ES (voir figure 2.2), la pertinence de l'utilisation d'outils informatiques, tels que les tableurs, ainsi que la mise en place d'algorithmes est mise en avant. En effet, il est précisé que les notions et thèmes abordés tout au long de l'année se prêtent bien à de telles applications. Cependant, nous pouvons remarquer en parcourant le document qu'il n'y a que peu d'indications concernant les algorithmes qui peuvent être utilisés. De plus, d'autres points de matière que ceux que nous avons présentés ici peuvent faire l'objet d'une application algorithmique, citons notamment les matrices et les graphes. Le potentiel du programme n'est par conséquent pas totalement exploité.

Série S

Parmi les trois grandes parties du programme de terminale en filière scientifique (analyse, géométrie, statistiques et probabilités), l'algorithmique est absente de celle qui concerne la géométrie. Nous présentons donc dans la suite les deux autres parties.

Analyse

À nouveau, l'algorithmique est fortement présente dans la partie concernant les suites. Premièrement,

> Dans le cas d'une limite infinie, étant donnés une suite croissante (u_n) et un nombre réel A, déterminer à l'aide d'un algorithme un rang à partir duquel u_n est supérieur à A. [MENJ, 2011b]

Il s'agit d'un exemple similaire à celui présenté précédemment pour la série ES-L, où l'algorithme n'est utilisé que comme un outil pour résoudre un problème en particulier, ici la détermination du seuil. Deuxièmement, il est précisé que des activités algorithmiques sont menées dans le cadre de l'étude des suites, sans précision supplémentaire. Troisièmement, dans le cadre de la continuité sur un intervalle et du théorème des valeurs intermédiaires, un commentaire précise lui aussi que des activités algorithmiques sont à prévoir mais cette fois-ci la proposition est plus précise. En effet, il est ajouté que ces dernières doivent se faire dans le cadre de la recherche de solutions de l'équation $f(x) = k$. Il n'est cependant toujours pas expliqué quelles sont les activités qui sont attendues et nous pouvons sans mal imaginer que l'algorithme jouera à nouveau un rôle d'outil.

Un deuxième contenu de la partie analyse est mis en avant au niveau algorithmique, il s'agit de l'intégration. Il est attendu des élèves qu'ils soient capables, pour une fonction monotone positive, de créer un algorithme qui puisse déterminer un encadrement d'une intégrale. Le problème qu'il faut résoudre est clairement exposé. La capacité attendue quant-à-elle est à nouveau de "mettre en œuvre" un algorithme, ce qui revient à décrire et implémenter une méthode.

Statistiques et probabilités

Dans cette troisième partie du programme de terminale en section scientifique, la colonne commentaire des notions de conditionnement et d'indépendance propose que des activités algorithmiques soient effectuées (par exemple la simulation d'une marche aléatoire). L'interprétation de ce commentaire est la même que pour la partie correspondante dans les autres programmes, c'est-à-dire des exercices de simulations et d'utilisation de programmes dans une démarche de type expérimentale.

Enseignement de spécialité

De la même façon que pour le programme de terminale ES-L, la section du programme concernant l'enseignement de spécialité de la filière scientifique indique que les notions qui sont abordées se prêtent particulièrement à des applications algorithmiques, ainsi qu'à l'utilisation d'outils tels que les tableurs et les logiciels de calcul. Il n'y a aucune précision supplémentaire.

2.3 Spécialité ISN en terminale S

Après avoir analysé les programmes de la seconde à la terminale pour les filières classiques, nous pouvons nous pencher sur le cas plus spécifique de l'enseignement de spécialité de l'informatique et des sciences du numérique, dispensé en classe de terminale scientifique. Cet enseignement est une option que les élèves de la filière S peuvent choisir, au même titre qu'un enseignement de spécialité en mathématiques comme nous l'avons vu plus haut.

Ce cours de spécialité est composé de quatre thèmes distincts : « représentation de l'information », « algorithmique », « langages et programmation » et « architectures matérielles » [MENJ, 2017]. Au vu du but de notre étude exploratoire, nous allons nous contenter d'analyser la partie concernant l'algorithmique, qui est d'ailleurs bien distincte de la partie programmation. Rappelons d'ailleurs que le programme de ce cours a été modifié récemment, il ne se présente donc pas sous la même forme que les autres programmes que nous avons considérés. Pour rappel, les citations provenant de ce programme seront indiquées entre guillemets dans le corps du texte ou avec une indentation plus forte lorsqu'elles sont plus longues.

Dans l'analyse de ce document, nous allons par ailleurs distinguer la partie du document qui discute de l'algorithmique du contenu du programme. Pour plus de facilité, nous mettons à disposition l'extrait, relativement court, qui nous intéresse à la figure 2.3.

2.3.1 Préambule

L'introduction de la partie sur l'algorithmique débute par une définition de la notion d'algorithme. Il est écrit que

> Un algorithme se définit comme une méthode opérationnelle permettant de résoudre, en un nombre fini d'étapes clairement spécifiées, toutes les instances

5.2. Algorithmique

Un algorithme se définit comme une méthode opérationnelle permettant de résoudre, en un nombre fini d'étapes clairement spécifiées, toutes les instances d'un problème donné. Cette méthode peut être exécutée par une machine ou par une personne.

Les élèves ont déjà appris au collège à écrire, mettre au point et exécuter un programme. Les programmes de mathématiques des classes de seconde et première développent une pratique de l'algorithmique sur laquelle il convient également de s'appuyer.

A partir du développement d'algorithmes, l'élève s'initie à la notion de complexité algorithmique. Ces algorithmes sont exprimés dans un langage de programmation et exécutés sur une machine ou bien définis de manière informelle.

SAVOIRS	CAPACITÉS	OBSERVATIONS
Algorithmes de référence – recherche dichotomique ; – addition de deux entiers exprimés en binaire ; – tri par sélection ; – tri par fusion ; – recherche d'un chemin dans un graphe par un parcours en largeur ou en profondeur.	**Comprendre** un algorithme et **expliquer** ce qu'il fait. **Modifier** un algorithme existant pour obtenir un résultat différent. **Concevoir** un algorithme. **Programmer** un algorithme. **S'interroger** sur l'efficacité d'un algorithme.	On présente simultanément les notions d'algorithme et de programme, puis on les distingue. L'objectif est une compréhension de ces algorithmes et la capacité à les mettre en œuvre. Les situations produisant une erreur (division par zéro, dépassement de capacité) sont mises en évidence. On présente les complexités logarithmique, linéaire et quadratique sur les exemples de la recherche dichotomique, de l'addition de deux entiers et du tri par sélection.
Traitement d'image Programmation d'algorithmes simples sur les images bitmap.	Modifier format, taille, contraste ou luminance d'images numériques. Détecter des informations spécifiques.	L'objectif est d'appliquer effectivement des programmes simples à des images. On peut aussi étudier le floutage, la rotation, la recherche de contour, etc.

FIGURE 2.3 – Extrait du programme de cours de la spécialité ISN

> d'un problème donné. Cette méthode peut être exécutée par une machine ou par une personne.

Dans cette définition nous pouvons voir une distinction entre deux aspects relatifs à un algorithme : l'aspect effectif et l'aspect problème.

L'aspect effectif de l'algorithme est décrit par le caractère systématique de ce dernier, en soulignant sa finitude (le fait que l'algorithme se finisse en un nombre fini d'étapes) et sa non-ambiguïté (les étapes doivent être clairement spécifiées). Il est aussi précisé que son exécution doit pouvoir se faire par un opérateur, qui peut alors être une machine ou un individu.

L'aspect problème quant à lui est mis en évidence par la capacité de l'algorithme à résoudre « toutes les instances d'un problème donné ». Autrement dit, l'algorithme doit pouvoir produire une sortie qui correspond à chacune des instances du problème, ce qui fait référence à des entrées différentes.

Le deuxième paragraphe nous rappelle que les élèves ont déjà été en contact avec des algorithmes dans les années précédentes de leur cursus scolaire (notamment au collège comme nous l'avons montré dans la section dédiée). C'est effectivement au cours de ces années antérieures (collège, seconde et première) qu'ils ont appris à écrire, mettre en place et exécuter des programmes plus ou moins complexes. Le document préconise de ne pas laisser ces apprentissages de côté. En effet, « les programmes de mathématiques des classes de seconde et première développent une pratique de l'algorithmique sur laquelle il convient également de s'appuyer ». Enfin, le dernier paragraphe nous annonce que l'objectif pour la partie algorithmique de ce cours est d'aller plus loin dans la réflexion autour de l'algorithme. Il est question de la complexité des algorithmes que, comme nous

avons pu le constater dans les sections précédentes sur les programmes des cours de mathématiques, les élèves n'ont pas encore eu l'occasion de travailler. Il est aussi précisé que dans le cadre de cette spécialité, les algorithmes ne sont plus exprimés dans un premier temps en langage naturel, pour ensuite être traduits, mais plutôt exprimés directement en langage de programmation (toujours sans précision sur le langage à utiliser).

2.3.2 Programme

Le programme du cours de spécialité ISN est lui aussi subdivisé en trois composantes : les savoirs, les capacités associées à ces savoirs et les observations (qui ne sont autres que des commentaires indiquant la marche à suivre aux enseignants).

Dans les savoirs, nous retrouvons une liste d'algorithmes dits « de référence », mais aussi un second point qui concerne le traitement des images. Les algorithmes listés sont la recherche dichotomique, l'addition de deux entiers via leur expression en binaire, le tri par sélection ou par fusion et finalement la recherche d'une chemin dans un graphe. Les trois premiers algorithmes sont simples et sont du ressort du traitement de données et de la gestion des nombres en machine, leur problématique est donc assez basique et ils permettent aux élèves de comprendre comment sont gérées les données dans les machines actuelles.

Les capacités qui sont associées à ces savoirs sont les suivantes : la compréhension, l'explication, la modification, la conception, la programmation et l'interrogation. L'idée est que les élèves soient capables de comprendre un algorithme pré-existant, afin d'expliquer ce qu'il fait, mais aussi de le modifier pour pouvoir lui faire produire des résultats différents. Ensuite, il leur est demandé d'être capables de concevoir de toutes pièces un algorithme, probablement avec les indications du problème à résoudre ou dans le cadre de travaux autour d'une situation concrète. L'algorithme est utilisé ici en tant qu'outil de résolution de problèmes et la capacité d'interrogation n'apparait que pour la complexité de l'algorithme.

Dans les observations il est noté que les notions d'algorithme et de programme sont présentées en même temps mais qu'elles sont par la suite distinguées. Il s'agit là d'une première dans les programmes du lycée. En effet, jusque là nous n'avions jamais vu de réelle différenciation entre algorithmique et programmation tandis qu'ici le professeur doit clairement effectuer la distinction. Une autre remarque intéressante concerne les « situations produisant une erreur (division par zéro, dépassement de capacité) », il est expliqué que de telles situations doivent être explicitement mises en évidence. Ces problèmes relèvent des limites physiques du matériel et de la programmation en tant que telle. Par contre, aucun commentaire n'est fait sur le dernier algorithme (recherche de chemin dans un graphe) qui semble nécessiter des notions plus complexes telles que les graphes. Finalement, les complexités que le programme demande d'aborder sont les complexités logarithmique, linéaire et quadratique. Ces dernières doivent être observées sur base des exercices concernant la recherche dichotomique, l'addition de deux entiers et le tri par sélection.

Les savoirs énoncés pour le traitement d'image sont la programmation d'algorithmes simples sur des images bitmap. . Il s'agit d'un ajout vis à vis de la version précédente du programme, qui ne mentionnait pas ce type de savoirs. Les capacités attendues relèvent de la manipulation de base de ce type d'objets (les images numériques) : la modification du format, de la taille, du contraste ou de la luminance. Il est précisé que l'objectif de l'introduction d'une telle notion est d'appliquer des programmes simples à des images, chose qui n'avait jamais été demandée jusqu'alors, dans aucun autre cours. Cela permet d'enrichir les connaissances des élèves sur l'algorithmique et de fournir des résultats très visuels lors de l'application des programmes qu'ils ont pu écrire ou étudier. De plus, ces notions permettent, comme la case des observations l'indique, d'étudier d'autres caractéristiques des images telles que « le floutage, la rotation, la recherche de contour, etc ».

Globalement nous pouvons dire que dans ce programme, l'algorithme est une notion qui est vraiment nécessaire à la résolution des problèmes proposés. En effet, dans les autres programmes il était souvent question de mettre sous forme d'algorithmes des fonctions, des formules ou des suites (qui peuvent être utilisées sous leur forme initiale), tandis que pour ce cours, l'algorithme est essentiel dans la mesure où il est indispensable à la résolution des problèmes proposés.

2.4 Document « Ressources pour la classe de seconde »

En même temps que la parution du nouveau programme de seconde, un dossier d'accompagnement a été mis à la disposition des enseignants. Ce document intitulé « Ressources pour la classe de seconde — Algorithmique — » [MENJ, 2009b] a pour but d'aider les enseignants du lycée à se familiariser avec les notions d'algorithmique et de programmation. Les citations présentes dans cette section proviennent toutes de ce document ressource, elles sont insérées entre guillemets dans le texte lorsqu'elles sont courtes ou séparément et avec une indentation plus forte lorsqu'elles sont plus longues.

Rappelons par ailleurs qu'une bonne partie des enseignants n'a pas eu de formation à l'algorithmique et la programmation durant leurs études, raison pour laquelle un tel document de référence a vu le jour. En plus de conseiller les professeurs à travers ses six parties, le dossier propose des exemples de situations d'algorithmique à explorer en classe de seconde, ainsi qu'une présentation des logiciels utilisés. Dans cette section nous allons parcourir ce document afin de le résumer au mieux. Nous ne discuterons cependant pas de la dernière section qui présente très sommairement les différents langages utilisés pour les exemples d'algorithmes.

2.4.1 Présentation générale et initiation à l'algorithmique

L'activité algorithmique et sa place dans les programmes du lycée sont développées dans les deux premières parties du document ressource : « présentation générale » et « une initiation à l'algorithmique ». Analysons ces deux sections.

Dans la première partie, l'introduction des éléments d'algorithmique dans l'enseignement est motivée de la même façon que dans le rapport de la CREM [Kahane, 2000] : via

plusieurs arguments distincts. Le premier est que nous sommes susceptibles de rencontrer des algorithmes dans notre quotidien et que, comme nous l'avons vu dans la section concernant le collège, les élèves ont déjà eu l'occasion d'en manipuler durant leur scolarité. Le second relève par contre plus de l'omniprésence de la machine dans notre société et du fait qu'il serait intéressant de pouvoir mettre en évidence les algorithmes qui se cachent derrière ces objets, ces outils, que nous utilisons tous les jours et sur lesquels nous ne nous posons que rarement des questions. En effet,

> Leur présence cependant, ne se traduit pas par un contact direct avec l'utilisateur qui assimile volontiers « la machine » à son mode de fonctionnement.

Le troisième quant à lui se base sur la capacité de l'algorithmique à permettre d'aborder les objets mathématiques de façon différente :

> Le développement du calcul automatisé a permis (au niveau de la recherche) de développer de nouveaux objets (fractales...), de nouvelles méthodes de démonstration et a profondément modifié la pratique des chercheurs.

Il est ensuite rappelé qu'il faut garder la programmation à un niveau relativement simple puisque l'objectif n'est pas de former les élèves pour devenir programmeurs :

> (...) la classe de seconde est une classe de détermination et il ne s'agit pas d'y former des programmeurs mais de faire en sorte que les mathématiques et l'algorithmique soient au service d'activités de résolution de problèmes pour les sciences.

Enfin, quelques compétences à travailler sont données, telles que la compréhension du but d'un algorithmique pré-existant, la modification de ce dernier, son analyse complète (identification des données d'entrée, de sortie, ...) ou encore l'adaptation d'un algorithme aux contraintes du langage de programmation sélectionné.

La seconde partie quant à elle propose un résumé des éléments constitutifs d'un algorithme et des étapes qui le structurent (la préparation du traitement, le traitement et la sortie des résultats), le tout est ensuite illustré par un exemple (le lancer de deux dés). Pour cet exemple, les différents objets algorithmiques sont mis en évidence au fur et à mesure. Tout d'abord, il est question de l'affectation de valeurs à des variables et des structures alternatives du type « si, alors, sinon ». Ensuite, afin de pouvoir mettre en place d'autres structures possibles dans un algorithme, l'exemple est modifié en y insérant soit des boucles WHILE, soit des boucles FOR. La différence entre les deux types de boucles est alors expliqué comme étant le fait de connaitre à l'avance le nombre de lancers à effectuer ou pas. Pour finir, le document définit les entrées et sorties d'un algorithme comme respectivement la lecture et l'écriture des données.

Dans ces deux sections, l'algorithme est présenté comme étant un outil au service des mathématiques, associé à une démarche du type expérimental (vérification des résultats obtenus à la main par exemple) ou alors à une démarche de résolution de problèmes. Le document insiste aussi sur le fait que l'algorithmique ne doit en aucun cas faire l'objet d'un cours à part entière mais doit être insérée dans l'ensemble des chapitres de chaque année du programme de mathématiques. De plus, il est dit que l'introduction de nouvelles notions

d'algorithmique dans les différents chapitres du cours doit se faire de façon naturelle, c'est-à-dire lors de la « résolution de problèmes pour lesquels les démarches habituelles sont malcommodes ou peu performantes ». Par exemple, il est fait mention du cas de la répétition d'une tâche ou encore celui où un traitement « à la main » serait trop long et fastidieux. D'ailleurs,

> Ces situations peuvent être rencontrées lors de l'extension à des cas plus généraux de situations déjà rencontrées : recherche du pgcd de nombres très grands, tri d'un très grand nombre de valeurs numériques, simulations sur des échantillons de grande taille...

Nous pouvons déduire de ces exemples précis que l'algorithme n'est utilisé que pour sa capacité à effectuer une tâche un grand nombre de fois, c'est-à-dire comme un procédé systématique.

2.4.2 Exemples d'activités

Dans cette troisième partie du document ressource, de nombreux exemples d'activités algorithmiques sont proposées mais ils sont décrits de façon simple, uniquement dans le but de servir de base à l'introduction de ces exercices dans le cadre de l'enseignement de l'algorithmique. De plus,

> Toutes ces activités ont pour objectif la production ou l'interprétation d'algorithmes et leur vérification obtenue par l'action immédiate et pas nécessairement issue directement du champ mathématique.

Il est expliqué, dans l'introduction de cette section, que le langage utilisé dans ce type d'activités devra subir une évolution progressive : de la lourdeur des phrases du type « si tu es dans telle situation alors fais cela, sinon » à une écriture présentant une syntaxe précise pour les éléments algorithmiques récurrents tels que les boucles, les tests, etc... Une syntaxe de ce type permettra alors une transition plus naturelle vers les langages de programmation qui demandent un certain formalisme.

Les exemples qui sont proposés sont en grande majorité des jeux qui ont pour but de faire réfléchir les élèves à des algorithmes sans qu'ils s'en rendent compte et sans introduire, de prime abord, un formalisme un peu rébarbatif. Nous pouvons citer notamment le « jeu du cartable », une adaptation du problème du sac à dos à la vie quotidienne des élèves. Une autre partie des algorithmes listés concernent les automates, qui dépendent exclusivement d'algorithmes pour leur fonctionnement. Enfin, la dernière partie des propositions consiste en une activité de lecture d'algorithmes accompagnée d'un questionnement. Les élèves devraient répondre à des questions sur les algorithmes telles que « écrire une suite possible d'affichages » ou « donner un problème dont la solution est donnée par cet algorithme ».

2.4.3 Algorithmes et géométrie, fonctions et probabilités

Comme nous l'avons vu dans la section concernant la classe de seconde, la géométrie, les fonctions et les probabilités sont les trois grandes parties du cours de mathématiques en première année du lycée. Dans les trois dernières sections du document ressource, des

algorithmes sont à nouveau présentés en rapport avec les trois parties du programme de seconde. Ces derniers sont donnés à la fois en pseudo-code et en langage de programmation (dans le langage des calculatrices Casio, en Python, en Scratch, ...). Certains de ces algorithmes sont même ceux demandés dans le programme du cours de mathématiques de seconde.

Puisque nous nous intéressons ici au langage Python, nous présentons donc comme exemple un des algorithmes du document ressource [MENJ, 2009b] dont la traduction est proposée dans ce langage. Il s'agit de l'algorithme de distance entre deux points (voir algorithme 1 et figure 2.4).

De multiples algorithmes de ce type sont donc disponibles dans le document, ce qui permet aux enseignants de visualiser la façon dont ils doivent être présentés aux élèves. Le fait qu'ils soient fournis avec les commentaires dans le code facilite la compréhension dès la première lecture. De plus, la présence de multiples langages donne l'occasion aux professeurs de choisir celui avec lequel ils ont le plus d'affinité ou celui qui, pour eux, sera le plus adapté aux élèves. Néanmoins, l'enseignant qui choisit un langage de programmation en particulier devra être capable de traduire seul les autres problèmes présentés sous forme d'algorithme.

2.4.4 Bilan du document ressource

Ce que nous pouvons retirer de l'analyse de ce document est qu'il peut s'avérer être un bon outil de référence sur plusieurs points. En effet, il donne une définition assez précise de la notion d'algorithme ainsi que des exemples pertinents puisque la plupart sont directement issus des programmes de mathématiques du lycée. Les professeurs de seconde peuvent alors s'en servir comme base pour la modification ou la création de leurs cours, dans le but de correspondre aux attentes de la Direction générale de l'enseignement et de la scolarité. De plus, le document passe en revue les trois thèmes du cours de

Algorithme 1 : Algorithme de distance entre deux points

Données :
    xA, yA // Coordonnées de A
    xB, yB // Coordonnées de B
    D // Carré de la distance de A à B
Entrées :
    Saisir xA, yA, xB, yB
1 **début**
2 | D prend la valeur $(xB - xA)^2+(yB - yA)^2$
3 **fin**
Sorties :
    Afficher "AB^2 ="
    Afficher D
    Afficher "AB ="
    Afficher $\sqrt{D}$

```
# Calcul de la distance entre deux points
from math import * # permet d'utiliser sqrt (racine)

xa=input("Saisissez l'abscisse du point A :") #
ya=input("Saisissez l'ordonnée du point A :") # saisie des
xb=input("Saisissez l'abscisse du point B :") # coordonnées
yb=input("Saisissez l'ordonnée du point B :") # des points

D=(xb-xa)^2+(yb-ya)^2 # La puissance est notée ^
print 'AB^2=',D # Calcul exact (grand entier)
print 'AB=',sqrt(D) # Calcul approché
```

Figure 2.4 – Traduction en Python de l'Algorithme 1

mathématiques de seconde, il est donc assez complet. Cependant, comme le précise son titre, ce document vise les enseignants de seconde, or l'algorithmique est présente aussi en première et en terminale. Ce qui signifie que pour les deux autres années, les professeurs peuvent s'inspirer du document ressource mais ils ne disposent pas d'exemple d'activité algorithmique spécifique à la matière qu'ils doivent aborder avec leurs élèves.

Chapitre 3

Cadre théorique de la didactique des mathématiques

Dans la mesure où ce mémoire se place dans une démarche de recherche en didactique des mathématiques, il se doit d'être étayé de différents outils théoriques didactiques. Ces derniers serviront, dans le dernier chapitre, de support d'analyse d'une séquence de cours et proviennent de sources diverses issues de la didactique des mathématiques, dument mentionnées et citées. Le cadre théorique didactique ainsi exposé sera complété par un second cadre, plus spécifique à l'usage des TICE (Technologies de l'Information et de la Communication à destination de l'Enseignement) pour l'apprentissage.

Ce premier cadre théorique est composé d'éléments des théories de Yves Chevallard et de Guy Brousseau, respectivement la théorie anthropologique du didactique et la théorie des situations didactique. Les apports de ces auteurs étant aussi vastes que reconnus, il nous faut nous restreindre aux notions qui nous semblent pertinentes pour notre travail d'analyse futur. Nous ne cherchons donc pas à produire un résumé exhaustif de ces théories, mais bien à en extraire et à expliciter les composantes appropriées par rapport à notre objectif. Ces différents concepts sont exposés et contextualisés dans le cadre de la théorie à laquelle ils appartiennent mais aussi identifiés dans les travaux des autres chercheurs lorsque c'est possible. Précisons aussi que nous adoptons une vision constructiviste de l'apprentissage, que l'on doit notamment à Piaget et partagée par Brousseau.

3.1 Théorie anthropologique du didactique – Chevallard

Le premier point que nous allons aborder est celui de la **transposition didactique**. Issu de la théorie anthropologique du didactique de Yves Chevallard ([Chevallard, 1982] et [Chevallard et al., 1985]), le concept même de transposition didactique a évolué au fil des années, nous indique Nathalie Briant dans sa thèse, et nous allons le développer rapidement ci-dessous. Bien qu'il s'agisse de l'unique partie de la TAD de Chevallard que nous présenterons dans ce chapitre, il nous faut nous y attarder pour plusieurs raisons que nous exposerons en temps voulu.

3.1.1 Transposition didactique

La définition la plus simple que nous pouvons donner de la transposition didactique est celle du passage « du savoir savant au savoir enseigné » [Chevallard et al., 1985]. Par savoir savant, il faut comprendre le savoir reconnu et validé par la communauté scientifique, spécialisée en mathématiques par exemple. Ce savoir, lorsqu'il est destiné au milieu scolaire, subit un certain nombre de modifications avant d'être repris dans les programmes, écrit dans les manuels et proposé à des élèves en classe. La transposition didactique désigne ces différentes transformations que subit le savoir savant lors de sa transformation en savoir enseigné. Elle est composée de deux étapes consécutives : la **transposition externe** et la **transposition interne**.

La transposition externe correspond au passage du savoir savant au savoir à enseigner, celui décrit dans les textes officiels tels que les programmes et les référentiels. La transposition interne quant à elle a lieu dans la classe, elle fait référence à la transformation du savoir qui s'y effectue, c'est-à-dire à ce qui est réellement enseigné par le professeur. En résumé, le savoir savant se change en savoir à enseigner (transposition externe) puis en savoir enseigné (transposition interne). Chevallard s'arrête à la transmission du savoir par l'enseignant, mais nous pourrions considérer une étape supplémentaire qui ferait état du « processus d'apprentissage, d'appropriation, de construction des savoirs et des compétences dans l'esprit des élèves », comme le propose Perrenoud [Perrenoud, 1998]. L'ensemble de ces étapes constituent ainsi ce que Perrenoud appelle la **chaine de transposition didactique**, schématisée à la figure 3.1.

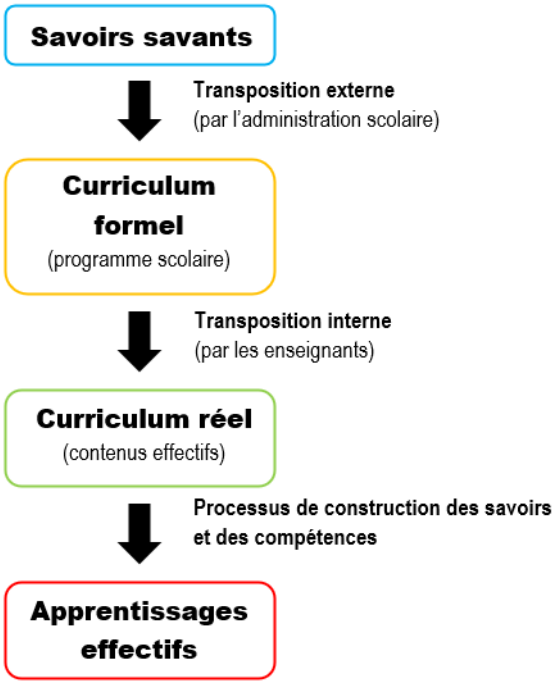


FIGURE 3.1 – La transposition didactique de Chevallard au sein de la chaine de transposition didactique. Adaptation de [Perrenoud, 1998].

Complétons cette définition de la transposition didactique par une citation de Chevallard, issue de son article « Pourquoi la transposition didactique ? » [Chevallard, 1982].

> Le concept de transposition didactique, par cela seulement qu'il renvoie au passage du savoir savant au savoir enseigné, donc à l'éventuelle, à l'obligatoire distance qui les sépare, témoigne de ce questionnement nécessaire, en même temps qu'il en est l'outil premier. Pour le didacticien, c'est un outil qui permet de prendre du recul, d'interroger les évidences, d'éroder les idées simples, de se déprendre de la familiarité trompeuse de son objet d'étude, bref, d'exercer sa vigilance épistémologique. [Chevallard, 1982]

Dans ce passage, l'auteur met en évidence la distance entre le savoir savant, le savoir à enseigner et le savoir enseigné. Il met en garde contre les transformations qui s'opèrent durant la transposition didactique. Par ailleurs, les modifications issues de la transposition externe sont le fruit d'une réflexion de la part de ce que Chevallard nomme la noosphère, que nous pouvons définir comme suit :

> La noosphère est donc l'ensemble des personnes qui pensent les contenus d'enseignement : les universitaires qui s'intéressent aux problèmes d'enseignement, les représentants du système d'enseignement (...), les auteurs de manuels, les inspecteurs scolaires, les représentants de la société (...) et les représentants du monde politique (...). [Clerc et al., 2006]

Dans notre étude future d'un scénario didactique, nous tâcherons d'identifier ces distances entre savoir savant, savoir à enseigner et savoir enseigné dans le domaine de l'algorithmique et de la programmation. Pour ce faire, nous reprendrons les contenus des programmes associés au cas particulier que nous rencontrerons, nous en analyserons les représentations qui y sont faites des objets algorithmiques et comment ceux-ci sont abordés dans la séquence. En amont de ce travail, nous définirons un autre cadre théorique particularisé à l'algorithmique.

3.2 Théorie des situations didactiques – Brousseau

Brousseau et sa théorie des situations didactiques paraissent incontournable lorsqu'on s'aventure un tant soit peu dans la didactique des mathématiques et, par extension, dans une optique d'analyse de séquence comme la nôtre. En effet, cette théorie « développe un cadre pour l'étude des situations d'enseignement des mathématiques » [Kuzniak, 2005].

Dans les pages qui suivent, nous allons considérer non pas l'ensemble des idées de Brousseau – comme nous l'avons précisé plus haut – mais deux grands aspects de sa théorie des situations : les situations didactiques et a-didactiques, ainsi que le contrat didactique. Ces derniers seront complétés de l'ensemble des notions qui y sont liées et qui nous paraissent pertinentes à aborder ici. Pour ce faire, nous nous basons non seulement sur les propos de Guy Brousseau lui-même, mais aussi sur le travail de différents auteurs qui se sont attelés à présenter sa théorie.

3.2.1 Situations didactiques

Comme son nom l'indique, la théorie des situations didactique se concentre sur ce que Brousseau appelle les « situations didactiques ». Précisons tout d'abord ce que désigne le mot « situation » :

> Une **situation** est l'ensemble des circonstances dans lesquelles une personne se trouve, et des relations qui l'unissent à son milieu. [Brousseau, 1997]

Le *milieu* dont il est question dans cet extrait est repris dans le glossaire de Brousseau comme « le système antagoniste de l'actant » [Brousseau, 2010].

Une **situation didactique** quant à elle est un cas particulier de situation dans laquelle l'intention d'enseigner est présente, que ce soit directement ou indirectement, mettant en scène l'élève en tant qu'actant face à un milieu sur lequel il agit et inversément. Brousseau les définit lui-même comme « des situations qui servent à enseigner » [Brousseau, 1997]. Dans le contexte d'une classe, on peut qualifier une situation de didactique « à chaque fois que l'on peut caractériser une intention d'enseignement d'un savoir par un professeur à un élève » [Johsua and Dupin, 1993]. Autrement dit, une telle situation englobe l'élève comme individu, ses relations avec le milieu dans lequel il évolue, ainsi que le professeur et ses intentions didactiques. Alain Kuzniak, qui se base sur les différents travaux de Guy Brousseau pour en présenter une introduction, propose la schématisation d'une situation didactique (voir figure 3.2).

3.2.2 Situations a-didactiques

Des ses recherches sur les situations, Brousseau s'est fortement intéressé aux situations qu'il nomme « **a-didactiques** ». Cependant cette appellation est parfois trompeuse car une situation a-didactique ne fait pas référence à une situation dénuée de toute intention

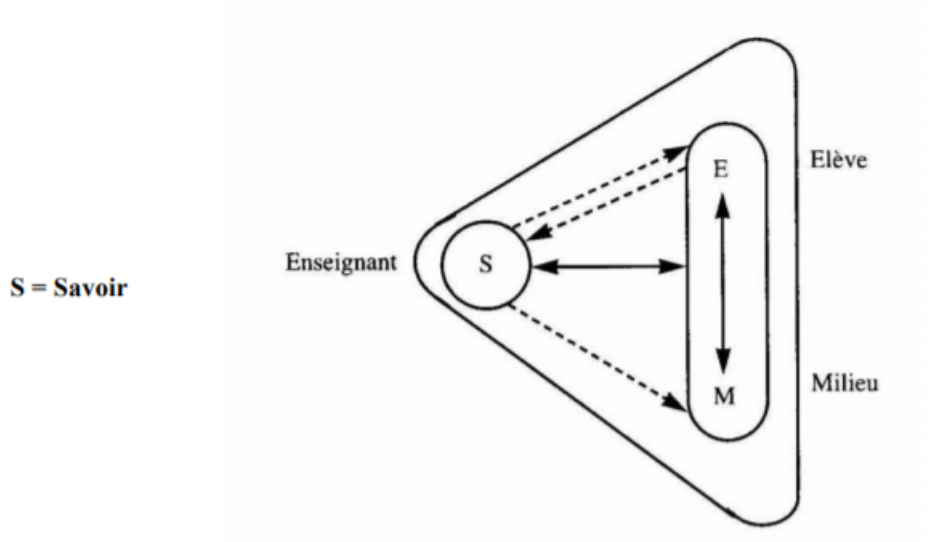


FIGURE 3.2 – Schéma d'une situation didactique selon Kuzniak [Kuzniak, 2005].

d'enseignement. Brousseau définit une situation a-didactique comme une situation dans laquelle

> Le maître se refuse à intervenir comme possesseur des connaissances qu'il veut voir apparaître. L'élève sait bien que le problème a été choisi pour lui faire acquérir une connaissance nouvelle mais il doit savoir aussi que cette connaissance est entièrement justifiée par la logique interne de la situation. [Brousseau, 1998]

Il est donc bel et bien question de l'enseignement d'un savoir et de l'acquisition de connaissances dans le cadre d'une situation a-didactique, tandis que le contraire correspond à une situation qu'on désigne alors comme « non didactique ». Alain Kuzniack, dans son ouvrage « La théorie des situations didactiques de Brousseau », indique qu'une situation non didactique

> est la situation rencontrée par le mathématicien ou l'usager des mathématiques lorsqu'il doit résoudre un problème dont la finalité première n'est pas l'apprentissage d'une quelconque notion mathématique. [Kuzniak, 2005]

On constate cependant que les situations a-didactiques mettent l'élève dans une position particulière par rapport au savoir en jeu. En effet, il lui est demandé de s'approprier la situation proposée par le professeur en adoptant le point de vue d'un « chercheur » dont l'objectif est la résolution du problème auquel il est confronté, et d'ignorer volontairement les intentions didactiques de l'enseignant. En provoquant et encadrant une telle situation, le professeur laisse aux élèves « la marge de manœuvre et d'initiative la plus grande possible » [Brousseau, 1997] durant le processus d'adaptation au milieu. Autrement dit, il laisse ainsi « jouer au maximum les mécanismes d'appropriation par les élèves du problème et de son dépassement » [Johsua and Dupin, 1993].

Nous l'aurons compris, les situations didactiques ont un but précis : amener les élèves à acquérir les connaissances et les savoirs visés et ce en les poussant à s'adapter au milieu avec lequel le professeur les met en relation. Le rôle de l'enseignant est de choisir judicieusement situations et problèmes afin de provoquer les adaptations voulues chez les apprenants, « provoquer la nécessité de la construction par l'élève de connaissances nouvelles » [Johsua and Dupin, 1993]. Une telle approche correspond à un apprentissage par adaptation – des élèves à la situation-problème – et est à distinguer de l'apprentissage dit formel. La différence majeure entre ces deux types d'apprentissage est la « présence et la fonctionnalité dans la situation didactique d'une étape de situation a-didactique » [Johsua and Dupin, 1993]. Ces deux notions de situation didactique et a-didactique sont donc intrinsèquement liées et constituent un pan important de notre cadre théorique didactique.

3.2.3 Différents types de situations a-didactiques

Brousseau a identifié trois catégories de situations a-didactiques : les situations d'action, de formulation et de preuve (ou de validation). Pour décrire ces situations, nous allons nous baser sur les définitions qui en sont données par Johsua et Dupin dans l'ouvrage « Introduction à la didactique des sciences et des mathématiques » [Johsua and Dupin, 1993] et les développer.

1. **Les situations d'action** où le problème appelle une action, un production de l'élève fondée sur un modèle implicite.

Dans une situation d'action, l'élève est confronté à un problème qu'il doit résoudre. Pour ce faire, il a recours à des actions sur le milieu qui, en retour, réagit à ces actions et renvoie des informations. Dans la définition ci-dessus, ce que l'auteur qualifie de *production de l'élève* peut être interprétée comme la suite des actions posées par l'apprenant sur le milieu. C'est en prenant en compte les informations renvoyées par le milieu et en adaptant ses actions par rapport à ces dernières que l'élève est amené à manifester ses connaissances.

Pour que l'élève puisse adapter ses actions, il lui faut juger les résultats de ces dernières et analyser les informations ainsi récoltées. En effet, si une certaine régularité apparait dans ces dernières, l'apprenant peut anticiper les futures réponses du milieu et ainsi adapter ses propres actions. Pour Brousseau, ce sont les connaissances qui permettent cette anticipation, et l'apprentissage est « le processus par lequel les connaissances se modifient » [Brousseau, 1997]. Ainsi, c'est au fil des échanges entre élève et milieu, entre actions et informations, que l'apprentissage a lieu. La « manifestation observable » [Brousseau, 1997] de cet apprentissage est expliquée par ce que Brousseau appelle un modèle implicite d'action. Le jeu d'interactions entre les deux camps dans une situation d'action est schématisé par Kuzniak et présenté à la figure 3.3.

2. **Les situations de formulation** qui permettent la mise en œuvre de modèles explicites, de langages, et où le savoir a une fonction de justification des actions et de contrôle.

Suite directe d'une situation d'action, la situation de formulation a pour but de transformer le modèle implicite que les élèves ont déduit lors de la phase précédente en un modèle explicite. Il s'agit donc d'une étape de justification des actions du modèle implicite au travers de leur formulation. Cette dernière peut prendre plusieurs formes en fonction des objectifs de l'enseignant, il peut s'agir de les communiquer à l'écrit ou à l'oral, en langage naturel ou mathématiques, au professeur ou à des camarades par exemple. Par ailleurs, Brousseau indique que la formulation du modèle implicite demande la mise en œuvre de répertoires linguistiques, de syntaxe ou de vocabulaire, qui par leur acquisition facilitent celle des connaissances qui sont exprimées à travers eux, « mais les processus sont distincts » [Brousseau, 1997].

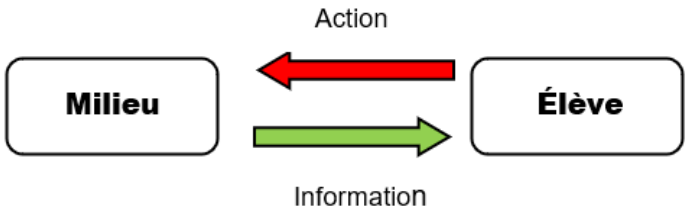


Figure 3.3 – Schéma d'une situation d'action. Adaptation de [Kuzniak, 2005].

3. **Les situations de validation** où sont mises en œuvre des mécanismes de preuve, et où le savoir a pour fonction d'établir ces preuves, de les contester ou de les rejeter.

Dans les situations d'action et de formulation, des corrections et des régulations empiriques, voire culturelles, permettent de « valider » les connaissances mobilisées – en jugeant de leur pertinence, de leur adéquation ou de leur conformité – mais il est encore possible de dépasser cette validation empirique et d'atteindre une validation sémantique et syntaxique des connaissances. C'est le rôle des situations de validation, aussi appelées situations de preuve.

Cette fois, l'élève n'est plus un informateur qui échange des informations, mais est un proposant dont le rôle est de convaincre un opposant. Le modèle implicite puis explicite considéré jusqu'alors doit être défendu, l'apprenant doit argumenter sur sa validité, donner des preuves de cette dernière. Pour ce faire, le proposant et l'opposant sont supposés posséder les mêmes informations et coopérer dans la recherche « du moyen de rattacher de façon sûre une connaissance à un champ de savoirs déjà établis » [Brousseau, 1997] (la connaissance dont il est question ici est celle construite dans les phases précédentes). En résumé, il est attendu des deux partis qu'ils prennent position, contestent et demandent la preuve des assertions d'autrui si survient le moindre désaccord. Brousseau explique que dans une situation de validation

> les élèves organisent des énoncés en démonstrations, construisent des théories – des ensembles d'énoncés de référence –, et apprennent comment convaincre sans céder aux arguments rhétoriques comme l'autorité, la séduction, l'amour propre, l'intimidation etc. [Brousseau, 1997]

3.2.4 Institutionnalisation

Afin de compléter les trois types de situations que nous venons d'aborder, il nous faut ajouter une situation dite d'institutionnalisation. La nécessité de cette dernière étape de la phase a-didactique d'une situation n'est pas apparue clairement à Brousseau dans un premier temps. Cependant, les résultats de ses expérimentations l'ont mené à réfléchir et à considérer l'institutionnalisation comme partie intégrante de l'acte didactique, cette dernière permettant de donner « à certaines connaissances le statut culturel indispensable de "savoirs" » [Brousseau, 1997].

> L'**institutionnalisation** est le passage pour une connaissance de son rôle de moyen de résolution d'une situation d'action, de formulation ou de preuve, à un nouveau rôle : celui de référence pour des utilisations futures, collectives ou personnelles. [Kuzniak, 2005]

En d'autres mots, c'est par le biais de l'institutionnalisation que l'enseignant indique aux élèves l'apprentissage qu'il cherchait à provoquer, les savoirs qu'ils doivent en retenir et qui peuvent être utilisés comme tels dans la suite. Brousseau indique que l'institutionnalisation peut aussi se traduire par l'appréciation de la production des élèves. En effet,

> Elle [l'institutionnalisation] peut consister en la reconnaissance par l'enseignant de la valeur d'une production des élèves. Elle affirme alors : (1) que la proposition de l'élève est valide et reconnue comme telle hors du contexte particulier de la situation présente, (2) qu'elle servira dans d'autres occasions, encore non connues, (3) qu'il sera alors plus avantageux de la reconnaître et de l'utiliser sous sa forme réduite que de l'établir à nouveau (4) qu'elle sera acceptée directement par tous ou au moins par les initiés. [Brousseau, 2010]

L'importance de l'institutionnalisation réside dans le fait que si cette dernière n'a pas lieu, les connaissances construites et accumulées par les élèves lors des trois situations précédentes – action, formulation et preuve – ne peuvent dépasser leur statut de connaissances individuelles et inévitablement contextualisées. Il faut cependant faire attention au moment où l'institutionnalisation est amenée. En effet, une institutionnalisation trop précoce a des effets néfastes sur l'apprentissage puisqu'elle interrompt le processus de création de sens, tandis qu'un recours trop tardif favorise les interprétations inexactes, ralentit l'apprentissage et gêne ses applications.

Nous sommes donc face à non pas trois mais quatre situations qui, mises bout à bout, constituent une bonne approche pour qui a pour but l'enseignement d'un savoir. En effet, selon Brousseau, « l'action, puis la formulation puis la validation culturelle et l'institutionnalisation semblent constituer un ordre raisonnable pour la construction des savoirs » [Brousseau, 1997]. Ce qui est illustré par A. Robert et al. comme suit :

> L'enseignant expose la consigne et présente un problème, les élèves cherchent le problème, individuellement ou en groupe, puis présentent leur résultats. Un débat entre élèves débouche sur de nouvelles questions, puis sur une extension des procédures utilisées et sur une synthèse où l'enseignant met en valeur les caractères importants du problème, les détache de leur contexte et les institutionnalise. [Robert et al., 1999]

Guy Brousseau admet cependant qu'il ne s'agit pas là d'une loi universelle, le processus peut différer et pour autant amener l'apprentissage visé.

3.2.5 Contrat didactique

Une autre notion centrale de la théorie des situations didactiques de Brousseau est celle de contrat didactique. Considérons tout d'abord la définition utilisée par A. Robert et al. dans leur publication « L'enseignement des mathématiques au lycée : un point de vue didactique ».

> Le **contrat didactique** est le résultat de la négociation des rapports établis explicitement et/ou implicitement entre un élève ou un groupe d'élèves, un certain milieu et un système éducatif, aux fins de faire approprier aux élèves un savoir constitué ou en voie de constitution.[Robert et al., 1999]

Nous pouvons aussi définir le contrat didactique comme l'ensemble des règles qui déterminent les rôles respectifs des élèves et du professeur au sein de la classe, ainsi que la répartition des responsabilités en ce qui concerne l'acte d'enseignement (sa prise en charge

et sa gestion) et le savoir. Les interactions entre le savoir, le professeur et les élèves, régulées par le contrat didactique, définissent une structure couramment appelée « triangle didactique » (voir figure 3.4).

Sur le terrain, les termes du contrat didactique se traduisent par les attitudes et les comportements du professeur, ses attentes envers les élèves mais aussi celles des élèves envers lui. Cela signifie que le contrat didactique est différent pour chaque couple maitre/-classe. Plusieurs dérives, paradoxes et dysfonctionnements de l'enseignement sont liés au contrat didactique et à ses effets, Brousseau en a identifié plusieurs (l'effet Topaze, l'effet Jourdan, etc) mais nous ne les aborderons pas ici.

Comme nous l'avons précisé, le contrat didactique est en majeure partie implicite et, en réalité, ne correspond pas à un véritable contrat, comme l'indique Guy Brousseau.

> Le contrat didactique n'est pas en fait un vrai contrat car il n'est pas explicite, ni librement consenti, et parce que ni les conditions de ruptures, ni les sanctions ne peuvent être données à l'avance puisque leur nature didactique, celle qui importe, dépend d'une connaissance encore inconnue des élèves. [Brousseau, 2010]

De plus, il n'est pas immuable puisqu'il s'adapte non seulement aux élèves mais aussi « au savoir en jeu, au moment et au type de tâche », comme le précise S. Johsua [Johsua and Dupin, 1993]. Ce dernier développe le caractère changeant du contrat en précisant que l'agencement des différentes phases d'une situation didactique exige des adaptations du contrat didactique dans la mesure où ce qui est attendu des élèves diffère pour chacune d'entre elles.

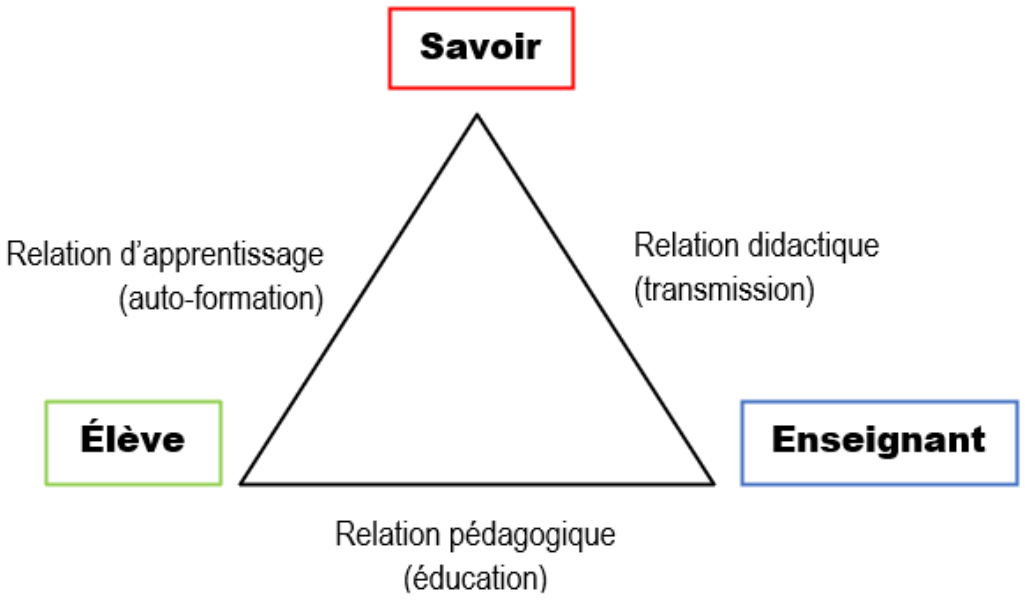


FIGURE 3.4 – Triangle didactique entre le savoir, l'enseignant et les élèves. Adaptation de [Henry, 2020].

> Dans la phase a-didactique d'une situation, l'élève est responsable des connaissances qu'il mobilise, des stratégies qu'il développe. Dans une situation d'action, de formulation ou de validation, sa responsabilité n'est pas engagée de la même manière. Par exemple, dans une situation de validation, il doit fournir des preuves, développer des arguments, réfuter des critiques alors qu'en situation de formulation, on lui demande de présenter sa solution, d'expliciter sa démarche. Il n'a pas à convaincre des contradicteurs. Lors de l'institutionnalisation des connaissances, le maître reprend le principal rôle dans la gestion des savoirs, il apparaît comme responsable officiel. [Johsua and Dupin, 1993]

La première raison pour laquelle nous considérons le contrat didactique comme une notion pertinente pour notre future analyserepose sur le fait que l'étude de son évolution permet d'en identifier ses ruptures. Dans son glossaire [Brousseau, 2010], Brousseau indique que ce sont les ruptures du contrat qui constituent la base de l'apprentissage, et pas le bon fonctionnement de celui-ci. Robert et al. reprennent cette idée comme suit :

> (...) il apparait nécessaire (...) de provoquer des ruptures [du contrat didactique] pour que le savoir en jeu ne soit pas dévoilé et reste à la charge de l'élève. Dans une perspective constructiviste, le traitement du savoir en situation de classe, va plutôt reposer sur les ruptures prévues du contrat. [Robert et al., 1999]

La seconde raison est que le contrat didactique permet d'expliquer le passage entre une situation didactique et une situation a-didactique, et ce via un phénomène que Brousseau nomme la dévolution.

> La **dévolution** est l'acte par lequel le professeur obtient que l'élève accepte, et peut accepter, d'agir dans une situation a-didactique. Il accepte les conséquences de ce transfert, en prenant le risque et la responsabilité de ses actes dans des conditions incertaines. [Kuzniak, 2005]

Par la dévolution l'élève prend et accepte la responsabilité de la résolution du problème auquel il est confronté dans une situation a-didactique. Elle est donc indispensable pour que l'apprenant construise les connaissances liées au savoir en jeu.

3.2.6 Variables didactiques et sauts informationnels

Lors de la construction d'une situation didactique, le professeur dispose d'une liberté toute relative en ce qui concerne les variables sur lesquelles il peut agir. Brousseau distingue les variables cognitives et les variables didactiques. Les premières sont telles qu'un changement dans leur valeur induit une modification dans les connaissances optimales liées à la situation, les secondes elles aussi des variables cognitives mais qui, cette fois, peuvent être fixées par l'enseignant.

> Une **variable didactique** est un élément de la situation qui peut être modifié par le maitre, et qui affecte la hiérarchie des stratégies de solutions (par le coût, la validité, la complexité). [Briand, 1991]

Pour illustrer ce concept, nous pouvons reprendre les exemples proposés par Johsua et al. dans leur livre « Introduction à la didactique des sciences et des mathématiques » [Johsua and Dupin, 1993]. L'âge des élèves ou leurs connaissances antérieures – qui ont une influence sur leur réussite d'un exercice – sont des variables sur lesquelles l'enseignant n'a aucune prise lors de la création d'une situation, il ne s'agit donc pas de variables didactiques. Par contre, dans un problème numérique, les nombres constituent des variables didactiques.

La définition précédente sous-entend que, pour un savoir visé, une situation peut demander de mobiliser des stratégies de résolution différentes en fonction des valeurs des variables didactiques qui la composent. Autrement dit, selon le choix de ces valeurs, une situation peut se complexifier fortement, induisant ainsi une modification des stratégies optimales qui lui sont associées mais aussi à « différentes manières de connaitre ce même savoir » [Brousseau, 1997]. En effet, rappelons-le, les élèves s'adaptent à la situation qui leur est présentée.

Lorsqu'un changement de stratégie est encouragé ou s'impose suite à la modification d'une variable didactique, on appelle cela un saut informationnel.

> On appelle **saut informationnel** un changement de valeur d'une variable didactique à l'intérieur d'une situation susceptible de provoquer un changement de stratégie. [Johsua and Dupin, 1993]

Un tel phénomène ne se produit que si la valeur de la variable didactique est assez modifiée pour que la stratégie précédente soit à présent inefficace. Chez Brousseau, cette progression « par sauts informationnels » trouve sa source dans la nature même de l'apprentissage.

> L'apprentissage par adaptation suppose qu'on choisisse les variables de façon à ce que la connaissance que l'on veut "faire découvrir" soit significativement plus avantageuse que toute autre. [Brousseau, 1997]

L'étude des variables didactiques d'une situation nous parait une approche pertinente pour notre analyse future d'une séquence de cours. En effet, cela permet d'identifier les différentes stratégies de résolution que les élèves sont susceptibles de déployer et de les caractériser en regard des objectifs visés. Ces différents intérêts sont appuyés par Johsua et Dupin dans le passage ci-dessous.

> Dans un travail d'analyse de document pédagogiques, d'outils pour le maitre, l'identification des principales variables didactiques des situations proposées est pertinente. [Johsua and Dupin, 1993]

Chapitre 4

Cadre didactique des TICE

Comme nous avons pu le constater dans notre état des lieux des programmes français au chapitre 2, l'algorithmique et la programmation dans l'enseignement, que ce soit dans un cours de mathématiques ou dans une option comme la spécialité ISN en terminale S, impliquent l'utilisation de technologies spécifiques telles que la calculatrice ou les ordinateurs munis de logiciels adaptés (comme les tableurs ou le logiciel Scratch pour le collège). Ces nouvelles technologies, que l'on appelle « Technologies de l'Information et de la Communication pour l'Enseignement » ou plus couramment « TICE », font en réalité partie intégrante de l'enseignement des mathématiques depuis de nombreuses années déjà et de multiples recherches ont eu pour objet l'étude de leur influence sur l'enseignement et l'apprentissage. Dans ce chapitre, nous tâchons de présenter les résultats de quelques-unes de ces études, et ce dans le but de définir un cadre théorique de référence spécifique aux TICE auquel nous pourrons nous référer par la suite. Nous appliquons aussi ces théories au cas précis des algorithmes et des programmes en guise de conclusion.

4.1 Genèse instrumentale – Rabardel

La première théorie que nous considérons dans notre cadre didactique des TICE est celle de Rabardel, exposée dans son ouvrage « Les hommes et les technologies : Approche cognitive des instruments contemporains » publié en 1995 [Rabardel, 1995]. Nous présentons ici les points centraux de cette théorie, ces derniers étant repris notamment dans de nombreux autres travaux, tels que ceux de Luc Trouche [Trouche, 2004] ou encore la thèse de Nathalie Briant [Briant, 2013] qui nous servent de références supplémentaires.

4.1.1 Artefacts et instruments

Dans une situation d'enseignement, une calculatrice ou un logiciel sont des **outils** qui jouent un rôle parfois déterminant dans la construction des savoirs. Contrairement à l'interprétation qui a pu en être faite dans le chapitre précédent, le terme outil est à prendre ici dans le sens que lui donne Pierre Rabardel, c'est-à-dire comme un objet matériel, que l'on nommera aussi **artefact**, auquel peut avoir recours un **sujet** [Rabardel, 1995]. Lorsqu'il utilise un artefact, le sujet se crée ses propres représentations personnelles de la façon dont il fonctionne, ses possibilités, ses limites, les contraintes liées à son utilisation,

etc. Ces conceptions nouvelles autour de l'outil, façonnées par le sujet, en font ce que Rabardel appelle un **instrument**.

> La distinction entre artefact et instrument réside dans les conceptions associées. Si l'artefact est premièrement conçu et réalisé par une personne ou une équipe de personnes pour répondre à un (des) objectif(s) précis, l'instrument est construit par le sujet à partir de cet artefact au cours de son usage lors d'une activité. [Claver, 2013]

En d'autres termes, un instrument est « une entité mixte qui tient à la fois du sujet et de l'artefact » [Rabardel, 1995]. Il est constitué d'une composante matérielle, qui n'est autre que l'outil, et d'une composante relative au sujet que l'on nomme **schème d'utilisation**.

> L'instrument est alors l'unité entre un artefact et l'organisation d'actions possibles, appelées les schèmes d'utilisation, qui constituent un ensemble structuré d'invariants correspondant à des catégories d'opérations réalisables à l'aide de l'artefact considéré. [Briant, 2013]

Ces schèmes d'utilisation peuvent être le fruit d'une construction propre du sujet, ou être le résultat d'appropriation de schèmes sociaux d'utilisation, précise Rabardel, c'est-à-dire qui ont vu le jour dans la collectivité.

4.1.2 Genèse instrumentale

Le processus par lequel un artefact se transforme en instrument, que nous avons déjà entrevu plus tôt, est appelé **genèse instrumentale**. Rabardel lui distingue deux dimensions en interaction : l'une qui concerne le sujet – l'instrumentation –, l'autre qui concerne l'outil – l'instrumentalisation – et qui sont définies comme suit et représentées à la figure 4.1.

- Les processus d'**instrumentation** sont relatifs à l'émergence et à l'évolution des schèmes d'utilisation et d'action instrumentée : leur constitution, leur fonctionnement, leur évolution par accommodation, coordination combinaison, inclusion et assimilation réciproque, l'assimilation d'artefacts nouveaux à des schèmes déjà constitués etc.
- Les processus d'**instrumentalisation** concernent l'émergence et l'évolution des composantes artefact de l'instrument : sélection, regroupement, production et institution de fonctions, détournements et catachrèses [1], attribution de propriétés, transformation de l'artefact (...) [Rabardel, 1995]

Plus simplement, lors de l'instrumentation, l'artefact impose ses contraintes au sujet, l'obligeant ainsi à s'y adapter et conditionnant ses actions, ses schèmes d'utilisation. Lors de l'instrumentalisation par contre, c'est le sujet qui adapte l'outil à l'usage qu'il souhaite en faire, à ses « habitudes de travail » [Trouche, 2004] et ses objectifs. Il se sert de ses connaissances pour déterminer les fonctionnalités de l'outil qui vont lui permettre d'atteindre son but, même si l'utilisation qui en résulte ne correspond pas à celle qui avait

1. La catachrèse désigne ici la création par le sujet de nouvelles fonctions de l'artefact qui n'étaient pas prévues initialement, Rabardel la définit comme « un concept qui désigne l'écart entre le prévu et le réel dans l'utilisation des artefacts » [Rabardel, 1995]

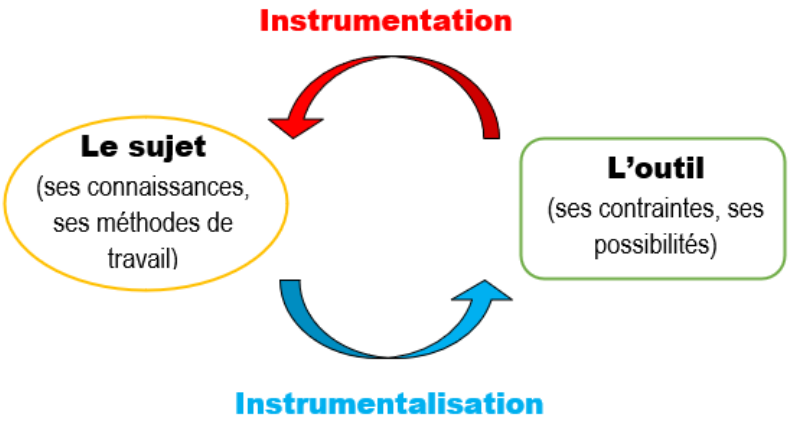


Figure 4.1 – Illustration de l'instrumentalisation et l'instrumentation de Rabardel. Adaptation de Luc Trouche [Trouche, 2004].

été imaginée par les créateurs de l'outil au départ. Luc Trouche illustre ces mécaniques en envisageant une calculatrice graphique comme artefact.

> Les processus d'instrumentalisation sont des processus de différentiation des outils eux-mêmes. Ils peuvent apparaître parfois comme un processus de détournement de l'outil : l'élève par exemple stocke des dessins ou des programmes de jeux dans sa calculatrice, modifie la barre des menus, enregistre des listes de théorèmes, etc. Mais ce processus peut aussi être compris comme une contribution des usagers au processus même de conception des outils et intégré ainsi par le maître dans la conduite des instruments de la classe (en montrant comment organiser le stockage des théorèmes dans la calculatrice, comment utiliser les logiciels de dessin pour traiter certains problèmes de géométrie, etc.). [Trouche, 2004]

La genèse instrumentale et les processus qui la composent sont à prendre en compte lorsqu'on poursuit un but d'apprentissage. En effet, la façon dont les élèves vont s'approprier les outils dans une activité, les connaissances qu'ils vont ainsi construire et les schèmes d'utilisation qu'ils vont mettre en place ont une influence sur leur apprentissage, sur la construction des savoirs visés. L'instrumentation nécessite une connaissance des « contraintes et des potentialités d'un outil donné » [Trouche, 2004]. Cela signifie que, lors de l'élaboration de situations didactiques, l'enseignant doit prendre en compte ces contraintes liées, dans le contexte d'outils informatiques, aux choix des concepteurs de ces outils et à la **transposition informatique**.

4.2 Transposition informatique – Balacheff

En complément de la théorie de Rabardel, nous allons nous pencher sur celle de Nicolas Balacheff, un chercheur qui s'est notamment interrogé sur les effets des TICE sur les savoirs à enseigner. Ces derniers subissent des modifications lorsqu'ils sont exploités au travers d'un environnement informatique, Balacheff nomme cela la transposition informatique, en

référence à la transposition didactique dont nous avons discuté dans le chapitre précédent. Dans cette section nous ne nous concentrons donc plus sur les élèves et leurs relations avec les technologies, mais bien sur les savoirs eux-même, ainsi que sur les caractéristiques des dispositifs informatiques employés pour les atteindre.

4.2.1 Contraintes de la transposition informatique

C'est en se basant sur la transposition didactique de Chevallard – « processus par lequel un élément du savoir savant devient une connaissance à enseigner puis un objet d'enseignement » [Briant, 2013] – et en y mêlant les contraintes intrinsèques aux « environnements informatiques d'apprentissage » [Balacheff, 1993] que Balacheff a défini la transposition informatique. La distinction entre ces deux processus de transposition provient du fait que le savoir enseigné dans un environnement informatique diffère de celui enseigné dans une situation dite « classique » d'enseignement. La figure 4.2 présente un schéma illustrant les différents traitements subis par le savoir savant et la place de la transposition informatique dans ce mécanisme.

La transposition informatique impose sur le savoir des contraintes de l'ordre de la modélisation et de l'implémentation informatique, précise Balacheff. Plus précisément, ces dernières sont de deux types : les « contraintes de la modélisation computable » [Balacheff, 1993] et les contraintes logicielles et matérielles du dispositif informatique. En d'autres mots,

> Les premières portent sur la représentation et le traitement interne des savoirs dans la machine et les secondes sur la représentation et le traitement au niveau de l'interface, autrement dit ce qui est visible pour le sujet. [Briant, 2013]

4.2.2 Univers d'un dispositif informatique

Pour l'auteur, le dispositif informatique induisant le processus de transposition est composé de trois univers : l'univers interne, l'interface et l'univers externe. L'univers interne fait référence aux « divers composants électroniques dont l'articulation et la mise en oeuvre permettent le « fonctionnement » du dispositif informatique » [Balacheff, 1993]. Le

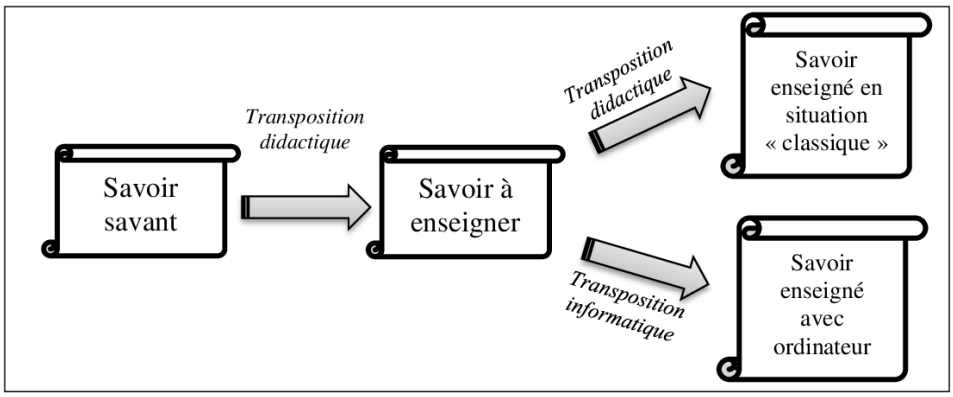


FIGURE 4.2 – Transposition didactique et informatique [Briant, 2013].

second, l'interface, est à prendre au sens de « lieu de la communication entre l'utilisateur humain et le dispositif informatique » [Balacheff, 1993]. Enfin, l'univers externe est celui dans lequel se trouve l'utilisateur, le sujet humain.

Pour illustrer ces trois univers, nous reprenons l'exemple du cercle proposé par Balacheff lui-même. Sur la figure 4.3, les trois univers sont mis en évidence. L'univers interne est caractérisé par des lignes de codes dans un langage de programmation donné, ce dernier imposant des contraintes sur la représentation de l'objet (de syntaxe, d'implémentation, ...). L'interface quant à elle affiche un cercle qui se trouve être composé de pixels en réalité. Balacheff indique que « suivant les caractéristiques de l'écran, le dessin du cercle produit par le dispositif informatique sera plus ou moins perçu comme un cerle au sens commun » [Balacheff, 1993]. Enfin, dans l'univers externe, « le cercle est une entité dont la nature est liée aux modes de représentation et de traitement disponibles, mais aussi aux classes de situations qui lui sont associées » [Balacheff, 1993].

4.3 Pseudo-transparence – Artigue

À présent nous nous intéressons au concept de pseudo-transparence de Michèle Artigue, enrichissant ainsi les apports de Balacheff et clôturant notre deuxième cadre théorique. C'est dans sa publication « Le logiciel "Derive" comme révélateur de phénomènes didactiques liés à l'utilisation d'environnements informatiques pour l'apprentissage » parue en 1997, que l'autrice développe le concept de pseudo-transparence. Elle la décrit comme un phénomène qui « renvoie à des décalages dans les modes de représentation (interne et à l'interface) des objets » [Artigue, 1997] dans un environnement informatisé, et insiste sur le fait que, malgré leur apparence négligeable, ces décalages ont des conséquences potentiellement néfastes sur le fonctionnement didactique.

En guise d'illustration de cette théorie, Briant teste dans sa thèse [Briant, 2013] l'écriture d'une expression algébrique dans les différents supports informatisés suivants : un tableau, une calculatrice scientifique et un logiciel destiné à l'apprentissage de l'arithmé-

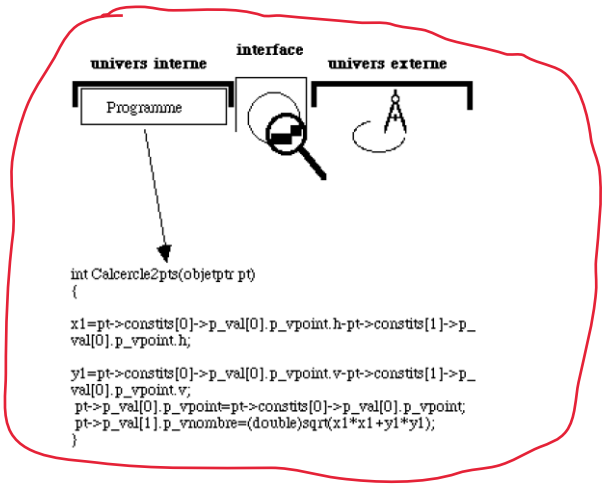


Figure 4.3 – Illustration par Balacheff des différents univers d'un dispositif informatique : exemple du cercle [Balacheff, 1993].

tique et l'algèbre. Pour une même expression algébrique à traiter, la fraction $\frac{x+4}{x-7}$, ces trois supports imposent une syntaxe différente. En effet, l'écriture dans un tableur est linéaire parenthésée, la fraction s'écrit $(x+4)/(x-7)$ et ne fait pas apparaitre la disposition spatiale de l'expression initiale retrouvée communément dans un environnement papier-crayon. La calculatrice quant à elle permet bien de faire apparaitre cette disposition spatiale mais nécessite un codage linéaire parenthésé au préalable, ce qui correspond alors à une écriture linéaire parenthésée spatiale en deux étapes : $(x+4)/(x-7) \rightarrow \frac{x+4}{x-7}$. Enfin, comme le montre Briant par une capture d'écran (reprise à la figure 4.4), le logiciel comporte un système de clavier virtuel permettant de compléter une forme pré-établie de quotient, l'expression est directement encodée dans sa forme spatiale. Briant analyse les résultats obtenus comme suit :

> Dans les deux premiers cas, il n'y a pas de transparence entre l'affichage de l'écriture de l'expression [algébrique] dans le tableur ou sur la calculatrice et la forme spatiale obtenue en environnement papier-crayon. Il est nécessaire d'adapter l'expression, de la transposer pour qu'elle soit comprise par la machine. [Briant, 2013]

Le manque de transparence entre les environnements informatisés et l'environnement papier-crayon se traduit donc ici par la nécessité de transformer l'expression lors du passage de l'un à l'autre. Ces différences entre représentations peuvent être interprétées en termes de distance entre les connaissances mises en œuvre dans chacun des environnements, indique Briant.

Artigue précise que, malgré le fait que les éléments évoquant un manque de transparence puissent paraitre problématiques au premier abord et que nous sommes tentés de les éliminer ou les éviter le plus possible, ce n'est plus réellement le cas lorsqu'on pousse l'analyse un peu plus loin. En effet, ces derniers « apparaissent aussi comme des éléments souvent mathématiquement ou didactiquement exploitables, à condition d'être acceptés et contrôlés » [Artigue, 1997]. Par exemple,

> La confrontation des deux environnements papier-crayon et informatique peut s'avérer bénéfique pour une compréhension approfondie de concepts, en exhibant des techniques ou des technologies différentes. [Briant, 2013]

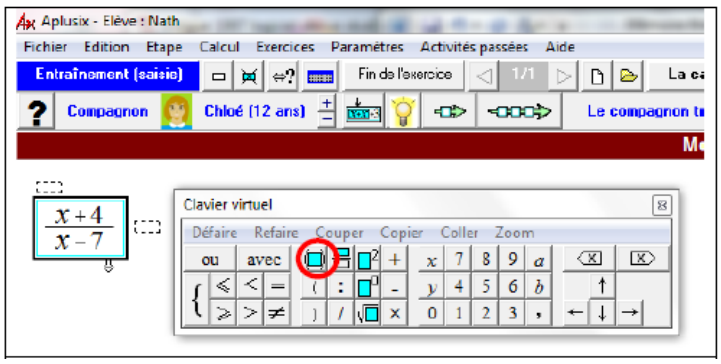


FIGURE 4.4 – Illustration du phénomène de pseudo-transparence : écriture spatiale sur le logiciel Aplusix [Briant, 2013].

Mettre les élèves dans une situation leur demandant d'effectuer une même tâche dans deux environnements différents, papier-crayon et informatisé, leur permettrait de se questionner sur les objets mathématiques manipulés et, à terme, développer une meilleure compréhension. Le rôle du professeur dans cette situation est important, en effet c'est à lui de faire ressortir les contrastes entre les représentations des objets mathématiques dans les différents environnements ou, au contraire, de les faire concorder, indique Briant.

> Haspekian (2005) indique alors le rôle important que va jouer l'enseignant qui, s'appuyant sur les différences des représentations mathématiques des objets dans les deux environnements, aura la charge de faire « coller » ces deux représentations ou au contraire de mettre en avant leurs différences, et de dégager ainsi les objets mathématiques visés. Cette mise en relation n'est pas seulement une source de difficultés supplémentaires à l'apprentissage d'un concept mathématique, mais peut permettre au contraire un enrichissement des concepts mathématiques en jeu. [Briant, 2013]

4.4 Application à l'algorithmique et la programmation

Contrairement aux théories que nous venons de présenter, ce dernier point est spécifique à la manipulation d'algorithmes telle qu'elle peut être envisagée dans un cours de mathématiques. En effet, il est question ici du passage d'un algorithme en langage mathématique à un programme informatique ; en passant par une écriture en pseudo-code. Autrement dit, la transition entre l'environnement papier-crayon et l'environnement informatisé. Nous nous en doutons, les transformations associées à ces différentes étapes ne sont pas anodines. Briant confirme nos doutes en présentant dans sa thèse ce qu'elle appelle la « double transposition », par analogie à la transposition informatique de Balacheff.

4.4.1 Exemple introductif

Pour introduire le concept de double transposition, nous allons reprendre ici l'exemple concret proposé par Briant [Briant, 2013]. L'idée est simple, l'exemple concerne un énoncé mathématique précis et nous allons ensemble parcourir les différentes étapes de transformation de ce dernier, tout d'abord en algorithme exécutable par un opérateur humain, ensuite en pseudo-code et enfin sous la forme d'un programme informatique.

L'énoncé mathématique considéré est le suivant : « un nombre entier positif n ($n \geq 5$) est premier s'il n'existe aucun nombre premier compris entre 2 et $\sqrt{n}$ qui divise n » [Briant, 2013], que l'on peut traduire comme une propriété permettant de dire si oui ou non, un entier naturel donné est un nombre premier. Bien qu'il s'agisse d'un énoncé n'étant à priori pas destiné à une implémentation en milieu informatisé, il peut tout à fait être transposé. L'algorithme ainsi obtenu, visible à la figure 4.5, est qualifié par Briant d'algorithme « mathématique » et est destiné à être exécuté par un opérateur humain dans un environnement papier-crayon.

À son tour, l'algorithme mathématique peut subir une réécriture dans un nouveau langage, permettant d'obtenir l'algorithme « informatisé » de la figure 4.6. Cet algorithme est

Algorithme n°1 (*mathématique*)

Soit $P_{\sqrt{n}}$, l'ensemble des nombres premiers de l'intervalle $[2 ; \sqrt{n}]$:

- S'il existe au moins une valeur k_0 de $P_{\sqrt{n}}$ telle que k_0 divise n, alors n n'est pas premier ;
- Sinon n est premier.

FIGURE 4.5 – Algorithme mathématique [Briant, 2013].

lui aussi destiné à un opérateur humain, néanmoins il est écrit en pseudo-code et plus en langage mathématiques. De plus, il s'agit d'une étape transitoire. En effet, un algorithme informatisé doit encore être écrit dans un langage de programmation pour pouvoir être exécuté par un ordinateur.

En observant les algorithmes des figures 4.5 et 4.6, nous pouvons constater que, malgré leurs similitudes, ils présentent aussi de nombreuses différences. Selon Briant, « la principale différence repose sur le destinataire de l'algorithme et sur le concept d'action élémentaire ». Rappelons-nous, les étapes d'un algorithme doivent être constituées d'actions élémentaires pour l'opérateur censé l'exécuter. Par conséquent, si l'opérateur est différent, les actions élémentaires le sont aussi. Briant identifie plusieurs points de divergence entre les deux algorithmes qui vont en ce sens.

1. Contrairement à une personne possédant certaines connaissances en mathématiques, un ordinateur ne dispose pas dans sa base de données, à priori, de la liste des nombres premiers existant entre 2 et $\sqrt{n}$. L'algorithme mathématique a donc dû être adapté à cette zone d'ombre lors de sa traduction en algorithme informatisé. De fait, il parcourt l'ensemble des entiers entre 2 et $\sqrt{n}$ au lieu de se baser uniquement sur les nombres premiers contenus dans cet ensemble.

2. L'expression « k divise n » a elle aussi dû subir quelques modifications dans le but d'être compréhensible pour l'ordinateur.

3. L'instruction « retourner » est apparue dans l'algorithme informatisé, ce qui révèle en réalité les prémisses de l'algorithme tel qu'il sera écrit dans un langage de programmation.

Il nous reste à présent à analyser la dernière étape, le passage de l'algorithme informatisé au programme informatique de la figure 4.7. Comme nous pouvons le constater, à l'instar de la transposition entre l'algorithme mathématique et l'algorithme informatisé, l'écriture sous la forme de programme demande un certain nombre d'adaptations et ces

Algorithme n°2 (*informatisé*)

- Pour chaque entier k, $2 \leq k \leq \sqrt{n}$, tester si $n \equiv 0[k]$;
- Si aucune valeur k ne divise n, alors retourner « n est premier » ;
- Sinon retourner « n n'est pas premier ».

FIGURE 4.6 – Algorithme informatisé [Briant, 2013].

dernières ne sont pas uniquement dues à la syntaxe du langage de programmation utilisé. En effet, elles proviennent aussi du passage des variables mathématiques aux variables informatiques, ainsi qu'à la nécessité d'introduire des instructions de lecture/d'écriture et de nouvelles variables informatiques nécessaires au bon fonctionnement du programme. Nous avons eu l'occasion de définir ces différentes notions dans un chapitre précédent. Notons par exemple l'apparition de la variable d dans le programme. Cette dernière n'était pas présente dans la précédente forme de l'algorithme simplement car elle n'était pas nécessaire à ce stade. Dans le programme, elle a un but précis, celui de « garder une trace d'un diviseur éventuel à la sortie de la boucle de test où sont recherchés les entiers de 2 à $\sqrt{n}$ divisant n » [Briant, 2013]. Autrement dit, elle permet que le programme se déroule correctement en prenant la valeur 1 lorsqu'un diviseur est détecté, ce qui signifie que n n'est pas premier.

Grâce à cet exemple, nous avons constaté que l'écriture d'un programme dans une perspective de résolution d'un problème mathématique nécessite de prendre en compte le fonctionnement même de l'ordinateur mais aussi d'identifier clairement quelles sont les instructions qu'il peut exécuter pour, en finalité, arriver à « faire effectivement "tourner" un programme avec un logiciel donné » [Briant, 2013]. En d'autres mots,

> L'apprenant en algorithmique va devoir apprendre à se décentrer de sa posture d'individu pour se placer dans la position de tenir compte de ce que sait faire la machine, s'il veut que son algorithme soit transférable en programme. [Briant, 2013]

Ce type de questionnement et de positionnement nous est familier à présent. De fait, tout cela font écho en réalité aux différents phénomènes que nous avons développés dans les pages précédentes : la transposition informatique de Balacheff, l'instrumentation de Rabardel et la pseudo-transparence d'Artigue.

```
1 VARIABLES
2 n EST_DU_TYPE NOMBRE
3 i EST_DU_TYPE NOMBRE
4 d EST_DU_TYPE NOMBRE
5 DEBUT_ALGORITHME
6 LIRE n
7 POUR i ALLANT_DE 2 A sqrt(n)
8 DEBUT_POUR
9 SI (n%i==0) ALORS
10 DEBUT_SI
11 d PREND_LA_VALEUR 1
12 FIN_SI
13 FIN_POUR
14 SI (d!=0) ALORS
15 DEBUT_SI
16 AFFICHER "n n'est pas premier "
17 FIN_SI
18 SINON
19 DEBUT_SINON
20 AFFICHER "n est premier"
21 FIN_SINON
22 FIN_ALGORITHME
```

Figure 4.7 – Programme informatique [Briant, 2013].

4.4.2 Double transposition

La double transposition que nous venons d'illustrer sur un exemple est schématisée à la figure 4.8. Sur cette dernière sont représentés trois cadres distincts, en interaction avec leur(s) voisin(s) ainsi que les deux transpositions. Nous allons expliciter ce schéma et son contenu en nous basant sur les propos de Nathalie Briant [Briant, 2013].

Pour un problème donné, le premier cadre représente la résolution pouvant en être faite dans le contexte « classique » des mathématiques, c'est-à-dire dans un environnement papier-crayon, avec l'aide de théories et de techniques mathématiques adaptées. Briant distingue cette résolution mathématique d'une résolution informatique. Dans l'exemple précédent, une telle résolution a pu mener à l'écriture d'un « algorithme mathématique », mais ce n'est pas toujours le cas. La première transposition a lieu suite à la résolution mathématique du problème et aboutit à un algorithme d'une nouvelle forme, un « algorithme informatisé », écrit en pseudo-code et cohérent avec les instructions que la machine peut effectuer. Enfin, la deuxième transposition se déroule lors du passage de l'algorithme informatisé au programme informatique. Programme dont l'écriture est cette fois-ci totalement adaptée à une exécution par une machine.

Le schéma met aussi en évidence les différents niveaux sur lesquels les deux transpositions agissent : le langage, les variables et les techniques/technologies utilisées. Pour le premier niveau, on passe du langage mathématique – « le langage utilisé usuellement par les écrits mathématiques » [Modeste, 2012] – au pseudo-code – « langage intermédiaire, inspiré des instructions des langages informatiques mais libéré de certaines contraintes et manipulant les objets mathématiques » [Modeste, 2012] – pour ensuite en arriver à un langage interprétable par la machine – un langage de programmation –. Pour le second niveau, on constate que les variables mathématiques partagées par la résolution mathématique et l'algorithme informatisé vont céder leur place aux variables informatiques au sein du programme informatique final. Enfin, pour le troisième niveau, ce sont les différentes

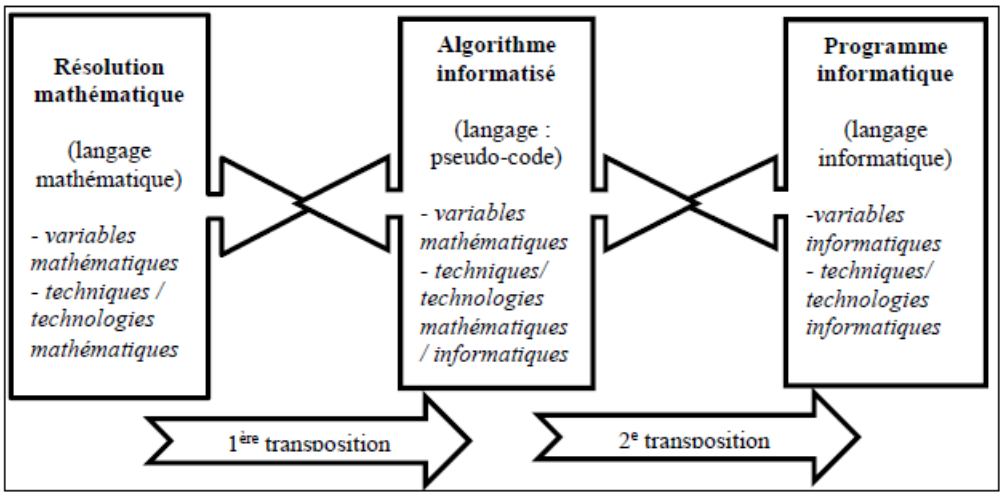


FIGURE 4.8 – Double transposition entre la résolution mathématique d'un problème et son implémentation en programme informatique [Briant, 2013].

techniques et technologies qui évoluent, comme nous avons pu nous en rendre compte dans l'exemple introductif.

En résumé, la double transposition ayant cours dans le type de tâche que nous considérons – l'écriture d'un programme informatique pour résoudre un problème – peut être perçue comme l'enchainement de deux phases :

> - la création d'algorithmes informatisés ou bien l'adaptation d'algorithmes mathématiques à partir de la résolution mathématique du problème en environnement « classique » papier-crayon.
> - l'adaptation de l'algorithme informatisé retenu en un programme, implanté sur un système informatique donné.
>
> [Briant, 2013]

Cependant ces phases ne doivent pas nécessairement être effectuées de façon successive, ni même chronologiquement. Briant ajoute même qu'il est possible que de multiples va-et-vient soient effectués avant de parvenir à un programme fini et fonctionnel, et ce même pour des professionnels qui, avec l'expérience, bénéficient de plus d'aisance dans la tâche.

4.4.3 Pseudo-transparence

Comme nous l'avons vu dans la section spécifique à la pseudo-transparence d'Artigue, l'utilisation de dispositifs informatisés n'est pas transparente pour l'utilisateur. Par extension, les transpositions que nous venons de développer ne le sont pas non plus. En effet,

> Une transposition n'est jamais transparente et un enseignement/apprentissage est nécessaire pour aborder les concepts de l'algorithmique. Ainsi, écrire un algorithme et le transformer en un programme informatique nécessite l'usage de variables informatiques et la connaissance de structures spécifiques et d'instructions (...). [Briant, 2013]

Pour étayer ses propos, Briant passe en revue les différentes structures et types d'instruction qui constituent un programme informatique et met en évidence la distance éventuelle qui sépare les notions informatiques de leur homologue mathématique. Ci-dessous nous reprenons brièvement les particularités ou difficultés que l'autrice a pu identifier pour chacune des structures et qu'il nous faudra garder en mémoire pour la suite.

Variables informatiques et variables mathématiques

Pour rappel, la notion de variable informatique est liée à celle de stockage de données dans la mémoire de la machine au cours de l'exécution d'un programme et sa valeur n'est pas fixe au cours du temps, elle peut être modifiée par l'affectation d'une nouvelle valeur à cette dernière.

Briant identifie deux difficultés en ce qui concerne les variables informatiques. Dans un premier temps, elle met en garde contre la similitude de vocabulaire entre variable

informatique et variable mathématique ainsi que la transposition informatique qui en découle. En effet, contrairement à une variable mathématique, une variable informatique « n'existe, à proprement parler, que lorsqu'elle est l'objet d'une instruction d'affectation » [Briant, 2013].

Dans un second temps, elle pointe du doigt la façon de représenter l'affectation d'une variable informatique, cette dernière pouvant induire en erreur. Modeste nous dit que l'affectation peut s'écrire de différentes façons :

> On écrit souvent « var :=val » ou « var ← val » pour affecter la valeur val à la variable var. Les langages de programmation utilisent parfois d'autres symboles pour la représenter, parfois même le symbole « = ». [Modeste, 2012]

Mais c'est justement l'utilisation du symbole de l'égalité qui peut causer des problèmes de compréhension, et à juste titre, puisque ce même symbole est utilisé en mathématiques pour signifier l'égalité entre deux valeurs ou deux variables. De plus, l'égalité mathématiques est une opération symétrique, ce qui n'est pas le cas de l'affectation.

La lecture et l'écriture

Contrairement aux variables informatiques, la lecture et l'écriture sont des objets informatiques ne possédant pas d'homologue mathématique, indique Briant. Par ailleurs, il n'est pas étonnant d'observer que les débutants en programmation présentent des difficultés à maitriser ces objets, à s'habituer à leur fonctionnement et à garder en tête que ces instructions sont exclusivement liées à la machine. De plus, « une confusion peut se produire entre les deux instructions, puisque la lecture consiste pour l'utilisateur à écrire au clavier et on utilise l'instruction écriture quand celui-ci doit lire sur l'écran » [Briant, 2013].

Les tests

Le point qui nous intéresse dans les structures de test n'est autre que la condition sur laquelle elles sont construites, plus particulièrement les opérateurs de comparaison qui y sont utilisés. Ces opérateurs n'ont pas le même statut en programmation qu'en mathématiques et un ordinateur n'est pas capable d'interpréter les relations d'ordre telles qu'elles sont présentées en mathématiques. Par exemple, Briant considère l'inégalité $2 < x < 4$ et nous explique que, malgré le fait que cette inégalité ait une signification mathématique, elle ne peut être utilisée sous cette forme en programmation. En effet, un ordinateur est incapable de comprendre cette condition composée de deux opérateurs de comparaison. Il est donc nécessaire de transformer la condition en utilisant un « opérateur logique » : AND, OR, XOR et NOT (et, ou inclusif, ou exclusif et négation en français). Ainsi, pour être compréhensible pour l'ordinateur, la condition ci-dessus doit s'écrire « Si x $>$ 2 et x $<$ 4 ». Il ne s'agit donc pas d'une simple nuance de notation mais d'une vraie différence liée au fonctionnement même de la machine.

> Cette subtilité montre la pseudo-transparence, au sens d'Artigue, des objets informatiques et mathématiques et nécessite la prise en compte des phénomènes

> liés à la transposition informatique (Balacheff, 1994) des savoirs mathématiques. Elle pose également la question de l'instrumentation (Rabardel, 1995). [Briant, 2013]

Les boucles

Enfin, en ce qui concerne les boucles, Briant indique que celles de type WHILE seraient moins facilement accessibles que les autres car elles nécessitent d'anticiper la condition d'arrêt de la boucle pour pouvoir la formuler, autrement dit « on doit anticiper la sortie de la boucle avant de répéter des actions » [Briant, 2013].

En résumé, nous venons de voir que l'ensemble des notions que nous avons abordées dans ce cadre didactique des TICE est aussi d'application à notre sujet, à savoir l'algorithmique et la programmation. Briant nous a permis de mettre un mot sur les différents phénomènes et processus qui ont lieu lors du passage d'une résolution mathématique d'un problème à un algorithme informatisé et de cet algorithme à un programme. Il s'agit de la double transposition, que nous aurons certainement l'occasion de voir à l'œuvre dans notre analyse de séquence.

Chapitre 5

Analyse à priori d'une séquence de cours

Dans ce dernier chapitre, nous allons effectuer une analyse à priori d'une séquence de cours que nous avons choisie. C'est ici que les différents cadres théoriques ayant attrait à l'algorithmique, la didactique et les TICE se recoupent et vont être exploités. Avant de donner quelques détails sur la séquence en elle-même, nous définissons les différentes hypothèses qui ont été prise en ce qui concerne notamment le contexte dans lequel nous nous plaçons ainsi que le public auquel la séquence est adressée. Ensuite, nous donnons de plus amples détails sur la séquence de cours en elle-même. Nous en donnons les sources, nous précisons plus en avant notre public, nous établissons les prérequis considérés ainsi que les objectifs d'apprentissage visés, nous discutons du matériel envisagé pour la séquence et nous donnons un résumé de son contenu. Pour finir nous analysons la séquence et nous en dégageons quelques perspectives.

5.1 Nos hypothèses

Avant d'effectuer l'analyse d'une séquence de cours, et pour que celle-ci soit pertinente, il nous a fallu poser quelques hypothèses relatives au contexte dans lequel nous nous plaçons. Ces dernières sont au nombre de quatre et découlent les unes des autres. Nous allons les présenter tout en présentant les raisons qui ont poussé à faire ces choix.

La première hypothèse, et non des moindres, est que nous imaginons que nous nous trouvons dans le cas de la France, plus précisément du lycée. Cela signifie que les élèves auxquels notre séquence est destinée ne sont pas novices en matière d'algorithmique et de programmation. En effet, proposer à des élèves du secondaire supérieur un cours portant sur ces matières, en Python de surcroit, sans qu'ils n'aient jamais eu l'occasion dans le passé de développer une pensée algorithmique, ainsi que les compétences qui y sont associées, n'aurait pas de sens. Cela ne correspondrait d'ailleurs pas aux conclusions du rapport publié par la CREM [Kahane, 2000] qui envisageait aussi l'évolution de la structure des enseignements au Collège en vue d'une introduction de l'algorithmique.

De notre première hypothèse découle la seconde, qui est que notre public dispose du même bagage de connaissances en algorithmique et en programmation que des élèves fran-

çais au sortir du Collège. Cela signifie qu'ils ont les capacités nécessaires pour « écrire, mettre au point (c'est à dire tester et corriger) et exécuter un programme simple » [MENJ, 2018], maitrisent des notions d'algorithmique et de programmation telles que les variables informatiques, le déclenchement d'une action par un événement, les séquences d'instructions, les boucles, les instructions conditionnelles, les boucles imbriquées ou encore la décomposition d'un problème en plusieurs sous-problèmes. De plus, ils sont capables de créer par eux-mêmes des algorithmes à partir d'une situation ou d'un problème donné.

Cependant, il n'existe aucune attente particulière en termes de représentation des algorithmes au Collège, ni même de langage de programmation ou de logiciel à utiliser pour atteindre les objectifs d'apprentissage. Par conséquent, nous émettons une troisième hypothèse, celle que les élèves ont eu l'occasion de manipuler des algorithmes en langage naturel et dans un langage de programmation simple tel que celui de Scratch.

Enfin, toujours dans le prolongement de la première hypothèse, notre quatrième supposition est que les élèves, une fois arrivés dans le secondaire supérieur, sont amenés à changer de paysage et à passer d'un environnement visant l'apprentissage de l'algorithmique, tel que Scratch, à un logiciel de programmation expert. Par conséquent, ils ne maitrisent pas le langage de programmation sur lequel nous nous basons dans ce travail, le Python. En outre, les élèves n'ont très certainement jamais décrit un algorithme en pseudo-code puisque le langage Scratch, par nature, ne pousse pas à l'écriture de pseudo-codes papiers.

5.2 Informations sur la séquence

Maintenant que les différentes hypothèses ont été exposées et que nous les avons en tête, nous pouvons nous concentrer sur la séquence proprement dite. Dans cette section, nous présentons les sources qui ont servi à l'élaboration de la séquence, le public auquel elle se rapporte, les prérequis supposés ainsi que les objectifs d'apprentissage du cours. Nous discutons aussi brièvement du matériel nécessaire à sa mise en œuvre et, enfin, nous proposons un résumé de la séquence.

5.2.1 Sources utilisées

La séquence de cours que nous allons analyser dans ce chapitre est très largement basée sur une séquence pré-existante, qu'il nous a donc fallu choisir au préalable. C'est dans ce but que nous nous sommes intéressé à ce qui a pu être proposé par nos voisins français de l' « Institut de Recherche sur l'Enseignement des Mathématiques » (IREM) de Paris. Porter notre dévolu sur les travaux d'une IREM n'est pas anodin, en effet, c'est nous assurer de partir d'une base cohérente et réfléchie pour la suite.

Sur le site officiel de l'IREM de Paris, nous retrouvons la page consacrée au groupe de recherche dédié à l'algorithmique. Cette dernière présente l'ensemble de leurs travaux et nous apprend que ce groupe a vus le jour suite à l'introduction de l'algorithmique et de la programmation dans les programmes du lycée en France, en 2009, et est constitué

« d'enseignants et formateurs du secondaire et d'enseignants-chercheurs en informatique » [Université de Paris, sd]. Le but des membres de ce groupe est de mener des réflexions sur l'introduction de l'algorithmique dans l'enseignement, mais aussi de proposer des formations à destination des professeurs du lycée, ainsi que ceux du collège par extension. Depuis leur création, le groupe a eu l'occasion de créer, soumettre puis tester diverses activités au sein de classes mais aussi d'organiser un certain nombre de stages et de formations.

Parmi toutes les ressources mises à disposition gratuitement sur ce site, nous avons sélectionné l'une des séquences faisant partie du stage le plus récent organisé par le groupe. Ce dernier, intitulé « Stage algorithmique et programmation au lycée », a eu lieu durant l'année scolaire 2018-2019 et se composait de différents exposés tels qu'une introduction aux notions d'algorithmique et à l'analyse d'algorithmes mais aussi de plusieurs activités en Python, réalisées lors la formation. Ces activités couvraient tant la géométrie que les probabilités et statistiques et l'analyse. C'est dans le domaine de l'analyse que nous avons trouvé le sujet pour notre séquence de cours : « la recherche du zéro d'une fonction réelle », disponible en annexe B [IREM de Paris, 2018].

Outre la séquence initiale provenant de l'IREM de Paris, il nous faut mentionner deux autres sources ayant servi à la création d'un aide-mémoire du Python. Ce dernier est basé sur le cours de « Programmation I » [Vanhoof, 2015] donné par Wim Vanhoof en première année de bachelier en sciences mathématiques à l'Université de Namur lors de l'année académique 2014-2015, ainsi que sur une introduction à Python proposée en cours de « Travaux Pratique de Programmation » [Blanchard and Dumont, 2019], donné en première année de Master en sciences mathématiques par Julien Blanchard et Morgane Dumont à l'Université de Namur lors de l'année académique 2018-2019.

5.2.2 Public visé

Dans nos hypothèses, nous avons mentionné que nous nous plaçons dans l'enseignement secondaire supérieur, mais il nous pouvons être plus précis que cela. En réalité, elle s'inscrit dans le cadre de l'enseignement des mathématiques au troisième degré du secondaire belge. Plus précisément, elle est adaptée à la cinquième année du secondaire, pour des élèves suivant une option incluant un cours de « mathématiques pour scientifique ». Dès lors, il s'agit de jeunes issus de l'enseignement général ou de technique de transition puisque ces deux orientations proposent le même programme de mathématiques.

C'est en observant le contenu des programmes de mathématiques de cinquième et de sixième année du secondaire que nous avons pu cibler notre public. Ce positionnement est directement lié à la séquence de l'IREM de Paris discutée précédemment. En effet, le contenu de cette séquence présente un certain nombre de prérequis mathématiques, d'aucuns uniquement abordés dans le programme de « mathématiques pour scientifique » du troisième degré. L'ensemble de ces prérequis mathématiques, ainsi que ceux que nous considérons en termes d'algorithmique et de programmation, sont repris dans la sous-section suivante.

5.2.3 Prérequis

La liste des connaissances préalables pour notre séquence est présentée ci-dessous. Ces prérequis sont extraits des processus présentés dans les « Compétences terminales et savoirs requis en mathématiques » [Wallonie-Bruxelles Enseignement, 2014] du cours de mathématiques pour scientifiques au troisième degré de transition, ainsi que des objectifs du cycle 4 du Collège pour l'algorithmiques et la programmation [MENJ, 2018].

Algorithmique et programmation

- Connaissances :
  - Notions d'algorithme et de programme (entrées-sorties, affectation, calculs, ...) ;
  - Notion de variable informatique ;
  - Déclenchement d'une action par un événement ;
  - Séquences d'instructions, boucles, instructions conditionnelles.
- Compétences associées :
  - Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné ;
  - Décrire des algorithmes en langage naturel ou en langage de programmation simple ;
  - Modifier un algorithme existant pour obtenir un résultat différent.

Mathématiques

- À partir de l'UAA2 - Suites :
  - Suites arithmétiques et géométriques ;
  - Convergence d'une suite.
- À partir de l'UAA3 - Asymptotes, limites et continuité :
  - Continuité ;
  - Théorème des valeurs intermédiaires ;
  - Méthode de dichotomie.
- À partir de l'UAA4 - Dérivée :
  - Tangente en un point du graphique d'une fonction ;
  - Dérivabilité ;
  - Fonction dérivée ;
  - Lien entre continuité et dérivabilité.

5.2.4 Objectifs d'apprentissage

Avant d'énoncer les différents objectifs d'apprentissage poursuivis par notre séquence, insistons sur le fait que notre public est déjà familier des concepts liés à l'algorithmique et la programmation. Par conséquent, nos objectifs ne visent pas l'acquisition de ces notions, que nous avons considérées comme des prérequis, mais plutôt leur mise en œuvre, notamment sous la forme de programme. Les énoncés des objectifs tels qu'ils sont présentés ci-dessous proviennent des programmes du lycée [MENJ, 2010b].

Dans le cadre de la séquence de cours, les élèves sont amenés à

- Décrire des algorithmes en pseudo-code ;
- Réaliser des algorithmes avec un logiciel adapté ;
- Interpréter des algorithmes plus complexes ;
- S'interroger sur l'efficacité d'un algorithme.
- Instructions élémentaires (affectation, calcul, entrée et sortie) :
  - Écrire en instructions élémentaires une formule permettant un calcul.
  - Écrire un programme calculant et donnant la valeur d'une fonction, ainsi que les instructions d'entrée et de sortie nécessaires au traitement.
- Instructions conditionnelles (boucles et itérateurs) :
  - Programmer un calcul itératif, le nombre de répétition étant donné.
  - Programmer une instruction conditionnelle, un calcul itératif avec une fin de boucle conditionnelle.

5.2.5 Matériel et logiciel

La description du matériel considéré pour notre séquence de cours n'est pas particulièrement longue. En effet, en termes d'outils, le professeur doit disposer au minimum d'un ordinateur et d'une solution de projection (vidéo-projecteur ou tableau blanc interactif) afin de pouvoir présenter plus aisément les différents codes et corrections. Les élèves quant à eux doivent disposer d'un ordinateur chacun, disposant d'un programme permettant l'écriture et l'exécution de codes en Python.

Pour l'élaboration de ce cours, plus particulièrement des différents codes en Python, nous nous sommes servis de l'outil « Anaconda ». Il s'agit d'une distribution Python, libre et open source, facilitant l'installation de Python et donnant accès à un grand nombre de bibliothèques pré-établies. Parmi ces bibliothèques nous pouvons citer le module `math`, qui inclut des fonctions mathématiques telles que le sinus et le cosinus par exemple.

Une fois installé, Anaconda permet entre autres l'utilisation d'un éditeur de code nommé Jupyter. Ce dernier s'ouvre et s'utilise dans le navigateur et, comme pour tout autre éditeur de code, c'est sur cette interface que l'utilisateur peut écrire et exécuter

ses codes en Python. Un fichier créé sur Jupyter, appelé un « notebook », se manipule aisément, nous pouvons supposer que les élèves devraient pouvoir appréhender cet environnement sans trop de difficulté. Cela dit, il existe bien d'autres éditeur de code permettant le même traitement, tel que Notepad++ ou encore Visual Studio Code.

5.2.6 Résumé

Notre séquence de cours est intitulée « Approximation d'un zéro d'une fonction réelle : approche algorithmique » [1] et est composée de trois parties distinctes, ainsi que d'un document complémentaire :

1. Mise en place,
2. Approche algorithmique et numérique,
3. Applications,
4. Complément : Aide-mémoire Python.

La première partie intitulée « Mise en place » permet, comme son nom l'indique, de mettre en place l'ensemble des outils relatifs à la programmation qui seront utiles dans la suite du cours. Cette étape introductive prépare les élèves aux manipulations qu'ils vont devoir effectuer en Python, le langage de programmation considéré dans le cadre de ce travail. De plus, elle permet d'amener le problème général sous-tenant l'ensemble de la séquence : déterminer une valeur approchée de la/les racine(s) recherchée(s) d'une fonction donnée.

La deuxième partie « Approche algorithmique et numérique » présente différentes méthodes algorithmiques permettant de répondre à la question posée par le problème que nous venons d'évoquer. Ces méthodes sont, dans l'ordre : la méthode de dichotomie, la méthode de Newton et la méthode de la sécante. Pour chacune d'entre elles, deux versions sont proposées : une version itérative et une version conditionnelle. Les élèves sont amenés, tout au long de la séquence, à explorer ces trois méthodes, à écrire en pseudo-code les algorithmes qui leurs sont associés (selon la version considérée) mais aussi à les implémenter en Python, les tester et comparer les résultats obtenus.

Dans la troisième partie, « Applications », deux exemples supplémentaires sont proposés. Ces exercices complémentaires mettent en scène les différentes méthodes algorithmiques de recherche de zéro d'une fonction sur des instances différentes, l'un portant sur une fonction polynomiale et l'autre sur une forme de la fonction de Bolzano.

Enfin, en complément du cours, il existe un guide de la programmation en Python, nommé « Aide-mémoire Python ». Ce dernier fait office de support à l'écriture des fonctions en Python et ne présente que la syntaxe des différentes instructions élémentaires telles que l'affectation simple, les structures conditionnelles et itératives, etc. Il contient aussi une liste non-exhaustive de fonctions Python pré-définies dont les élèves pourraient avoir besoin.

1. Le support de cours est disponible en annexe (en version destinée à l'enseignant et version destinée à l'élève)

5.3 Analyse à priori de la séquence

Nous allons à présent nous consacrer à l'analyse proprement dite de notre séquence de cours. Il s'agit d'une analyse à priori, cela signifie qu'elle est effectuée indépendamment de toute mise en œuvre réelle du cours et qu'elle se base sur des apports théoriques. Il n'est donc pas question de prédire ce qu'il pourrait se passer en classe mais bien de mieux appréhender les objectifs de la séquence et de comprendre comment les élèves vont réagir vis à vis de ces derniers. Pour cela, nous nous basons sur l'ensemble des informations que nous venons de donner à propos de la séquence ainsi que sur les cadres théoriques de l'algorithmique, de la didactique et des TICE que nous avons définis dans les chapitres précédents.

Afin de garantir une progression fluide dans les contenus, nous allons parcourir le document « Approximation d'un zéro d'une fonction réelle : approche algorithmique » de manière linéaire, partie par partie. Ce faisant, nous tâcherons pour chacune des situations rencontrées de mettre en évidence les notions d'algorithmique et de programmation en jeu, les différentes stratégies de résolution possibles, les connaissances préalables qui y sont associées ainsi que les difficultés que les élèves pourraient rencontrer, notamment en regard des différents phénomènes liés à l'environnement informatisé que nous considérons. Nous tâcherons aussi d'identifier les variables didactiques de ces situations, leurs modifications éventuelles et les conséquences de ces dernières. En ce qui concerne le professeur, nous présenterons sa place et ses actions au sein des différentes situations, par exemple en ce qui concerne les connaissances à institutionnaliser. Nous analyserons aussi l'évolution et les ruptures du contrat didactique.

Dans la suite, les citations, figures et algorithmes proviennent du support de cours de notre séquence version enseignant, fourni en annexe A. Pour ne pas surcharger les pages de notre analyse, nous n'y faisons apparaitre que les algorithmes et les codes qui servent notre propos, par conséquent il peut être utile de se référer à l'annexe de temps à autre afin de s'y repérer.

5.3.1 Mise en place

Comme nous l'avons vus dans le résumé, la première partie du document sert majoritairement à introduire des outils spécifiques à Python et au logiciel d'édition de texte que nous utilisons. Sa structure et son contenu sont très similaires à ceux de la séquence originelle, celle de l'IREM de Paris [IREM de Paris, 2018], mais il existe quelques divergences que nous expliciterons en temps voulu.

Pour cette section du cours ainsi que les suivantes, les élèves sont amenés à interagir avec un milieu bien particulier. En effet, comme nous l'avons précisé dans la partie « Matériel et logiciel », l'ensemble des situations que nous allons rencontrer nécessitent l'utilisation d'un dispositif informatisé, plus particulièrement d'un ordinateur muni d'un logiciel d'édition de code permettant la communication avec ce dernier. Par conséquent, le milieu dans lequel évolue chaque élève et avec qui il est mis en interaction est ici non seulement composé du professeur et des autres apprenants mais aussi de l'ordinateur.

Importer les modules, définition de la fonction sinus et représentation graphique

Dans cette première partie du cours, le positionnement du professeur par rapport aux élèves et aux savoirs en jeu ne change pas. En effet, il a ici pour but de guider les élèves au travers des différentes étapes de mise en place, et pour ce faire il conserve sa position de « détenteur du savoir ».

En tout premier lieu, les élèves doivent importer les libraires nécessaires au bon déroulement de la séquence. Il s'agit des libraires `math` et `matplotlib` qui permettent d'avoir accès à une série de fonctions en Python prédéfinies telles que les fonctions trigonométriques ou encore celles destinées à la représentation graphique de fonctions. L'importation de modules est caractéristique à Python et à son utilisation dans Anaconda, et ces manipulations préliminaires sont nécessaires mais ne font pas partie de nos objets d'apprentissage en tant que tels, nous en rediscuterons par la suite. Par contre, pour retrouver les instructions d'importation des deux modules, les élèves doivent recourir à l'aide-mémoire qui leur est communiqué. Le fait de se référer en temps utile à un outil comme cet aide-mémoire peut être considéré comme une compétence transversale qui sera travaillée tout au long de la séquence.

Ensuite, nous retrouvons la fonction mathématique sur laquelle se base une grande partie de la séquence. Il s'agit de la fonction sinus, présentée directement sous la forme d'une fonction Python `fsin` et que les élèves n'ont plus qu'à recopier dans leur notebook. La fonction choisie est une variable didactique de la situation pouvant éventuellement être modifiée ultérieurement. Dans tous les cas, le choix du sinus comme fonction de référence s'explique simplement : il s'agit d'une fonction courante bien connue des élèves, ce qui rend plus aisé la vérification des résultats obtenus par le biais des algorithmes dans la suite.

Une fois la fonction `fsin` encodée correctement, la séquence se poursuit sur une seconde fonction nommée `graphe` permettant d'afficher le graphe d'une fonction f sur l'intervalle $[a, b]$. À l'instar de la précédente, et toujours en suivant les propositions de la séquence de l'IREM, le code de cette fonction est donné et les élèves doivent le retranscrire sur l'ordinateur . Cependant, celui-ci est plus complexe à appréhender. En effet, il comporte un certain nombre de fonctions issues du module `matplotlib` (`plt.gca`, `plt.show`, etc) dont les élèves ne peuvent saisir le sens qu'au travers des commentaires insérés dans le code, et qui ne font d'ailleurs pas partie des apprentissages visés. Nous pouvons interpréter ces différentes fonctions comme des « boites noires », dans le sens où elles sont nécessaires pour obtenir le résultat escompté – afficher un graphe – mais l'utilisateur – l'élève – ne sait pas ce qui se cache derrière ces instructions, il n'a pas connaissance de leur fonctionnement. Il serait intéressant de réfléchir plus en avant sur la façon dont les élèves perçoivent ces boites noires, mais nous ne disposons pas de référence à ce sujet.

Jusqu'à présent nous n'avons pas modifié l'approche proposée par l'IREM, mais alors une question reste en suspens : comment expliquer la présence des codes complets des fonctions Python que les élèves n'ont plus qu'à retranscrire ? La justification nous la trouvons chez Rabardel. En effet, les modules ainsi que les fonctions `fsin` et `graphe` introduisent la

plupart du vocabulaire et des éléments inhérents à la programmation en Python, tel que l'importation de librairies de fonctions, les arguments d'entrée et de sortie, la syntaxe en blocs, l'importance des indentations ou encore les commentaires de code. Autrement dit, des éléments dont les élèves doivent avoir connaissance pour atteindre nos objectifs d'apprentissage mais qui sont de l'ordre de la syntaxe, des conventions avec lesquelles il faut composer et qui ne nécessitent pas d'être « construites » ou « découvertes ». Par ailleurs, en réécrivant ces codes, il est fort probable – voire inévitable – que certains élèves ne respectent pas la syntaxe au pied de la lettre, ou « à l'indentation près ». Nous pouvons dès lors nous attendre à des erreurs telles qu'un oubli de ponctuation, un manque d'indentation ou plus simplement une faute d'orthographe qui mettent en péril toute l'exécution du code. De plus, le notebook Jupyter étant doté d'un système de détection d'erreurs en cas de problème d'exécution, les élèves qui maitrisent convenablement l'anglais seront avertis du type et de l'emplacement de celles-ci, dans la mesure du possible. Les autres pourront se référer à l'enseignant pour identifier le problème et, si la situation l'exige, y remédier. C'est aussi dans cette optique que nous avons créé un aide-mémoire Python, c'est-à-dire pour que les élèves puisse disposer de toutes les informations nécessaires au travail sur nos objectifs d'apprentissage. En résumé, cela correspond au processus d'instrumentation, les apprenants sont informés et conscients des contraintes du langage qu'ils manipulent et doivent en adapter leurs actions. Il en est de même pour le logiciel utilisé, mais dans une moindre mesure puisqu'il suffit d'indiquer aux élèves comment exécuter les codes dans le notebook.

La remarque qui suit le code de la fonction `graphe` est quant à elle relative à la pseudo-transparence entre la représentation papier-crayon d'un concept mathématique et sa représentation en Python. En effet, elle met en évidence la distance qui sépare la représentation de la suite composée des nombres de la forme $a + \frac{1}{100}(b-a)$ en environnement papier crayon et celle qui en est faite en Python sous la forme de mutation de liste. En d'autres mots, la remarque introduite par l'IREM permet aux élèves de percevoir que derrière cette écriture particulière se dissimule en réalité un concept qu'ils connaissent déjà, ce qui leur permet de mieux appréhender le concept sous sa « nouvelle » forme.

Nous pouvons interpréter la question qui suit cette remarque d'une façon similaire puisque l'appel à la fonction `graphe` avec comme argument la fonction `fsin` et l'intervalle $[-4, 4]$ montre aux élèves qu'il ne s'agit en réalité que de la traduction dans un autre environnement d'une action qu'ils ont l'habitude d'effectuer en papier-crayon : tracer le graphe d'une fonction. Cela permet aussi de mettre en évidence la concordance entre la fonction `fsin` qui a été définie et la fonction trigonométrique sinus qu'ils connaissent et maitrisent, ce que Briant a identifié comme un des rôles du professeur lors de l'utilisation d'un environnement informatisé et que nous avons abordé dans notre chapitre 4. De plus, cette question révèle de façon explicite aux élèves un autre élément associé à Python : l'appel de fonction, que nous souhaitons qu'ils maitrisent au terme de cette séquence.

Une fois le bon graphe obtenu sur leur machine, il est demandé aux élèves de repérer sur celui-ci les différentes racines de la fonction sur l'intervalle restreint $[-\pi, \pi]$. Étant donné la fenêtre de représentation choisie pour le graphe, aucune ambigüité n'est possible en ce qui concerne les racines à identifier. Par ailleurs, la modification de la variable

didactique que constitue l'intervalle d'affichage influence la réponse des élèves dans le but qu'ils choisissent les valeurs exactes de ces racines, qui sont $-\pi$, 0 et π, sans arrondir la valeur de π à 3.14 par exemple. Ensuite, il est demandé aux élèves de déterminer ces mêmes racines de façon analytique et non graphique. Il est donc attendu qu'ils résolvent l'équation

$$\sin(x) = 0 \Leftrightarrow x = \pi + k\pi, \; k = \{-2, -1, 0\}. \tag{5.1}$$

Ces deux questions rappellent aux élèves la méthode pour déterminer la/les racine(s) d'une fonction f mais aussi, et surtout, font office de préparation au sous-point suivant qui réinvesti l'appel de fonction sur un intervalle différent, $[2, 4]$, et indique que la racine encadrée de la sorte est celle sur laquelle nous allons nous concentrer dans la suite.

Résolution exacte et approchée

Comme nous l'avons vu, les questions précédentes ont pour utilité première de soutenir le discours portant sur la résolution exacte et approchée d'une équation du type $f(x) = 0$. En faisant travailler les élèves sur la résolution exacte de l'équation 5.1, ces derniers se trouvent face à une résolution rapide et aisée, or c'est loin d'être toujours le cas. En effet, outre quelques exercices plus difficiles auxquels ils ont pu être confrontés dans le cadre du cours de mathématiques auparavant, il existe des situations où l'humain seul, armé de sa calculatrice, ne peut résoudre aussi simplement une équation de ce type ; nous aurons l'occasion de rencontrer un exemple très parlant de ce phénomène dans la toute dernière partie de ce cours. De plus, bien que certains théorèmes assurent l'existence et/ou l'unicité d'une solution à une telle équation, ils ne donnent pas d'information sur la valeur de cette solution. Ce dernier point de la section « Mise en place » présente aux élèves qu'il existe un moyen de « contourner » ces difficultés, que des méthodes adaptées peuvent être utilisées et que nous allons justement nous intéresser à certaines d'entre elle dans la suite de ce cours. Comme nous le savons, il s'agit en réalité de méthodes algorithmiques permettant de déterminer une valeur approchée de la/les racine(s) recherché(es) d'une fonction réelle donnée. Autrement dit, les différents algorithmes présents dans la suite proposent une solution à la question posée par ce problème. Nous verrons par ailleurs que ces algorithmes ne considère pas exactement la même famille d'instances. Somme toute, cette dernière partie justifie l'ensemble de la séquence auprès des élèves.

5.3.2 Approche algorithmique

Nous entrons à présent dans le vif du sujet. Dans cette deuxième section du cours, nous allons rencontrer les algorithmes en pseudo-code et les codes en Python associés à nos trois méthodes de recherche de zéro d'une fonction : la méthode de dichotomie, la méthode de Newton et la méthode de la sécante. Le fil conducteur de cette partie « Approche algorithmique » est le test des trois méthodes sur une seule et même fonction. Pour une question de facilité de vérification des résultats et de cohérence, la fonction considérée est la fonction sinus déjà utilisée dans la section précédente. Il en est d'ailleurs de même dans la version originale de la séquence [IREM de Paris, 2018]. Au travers de ces différentes applications, nous aurons l'occasion de comparer les méthodes entre elles, ainsi que leurs différentes versions (itérative ou conditionnelle) en termes de vitesse de convergence, plus

précisément en nombre d'itérations.

La progression dans ces méthodes suit une logique simple. En effet, la première méthode, celle de dichotomie est la plus générale des trois. Elle permet d'obtenir l'approximation d'une racine d'une fonction réelle en partant uniquement d'une hypothèse de continuité sur un intervalle donné. La seconde, la méthode de Newton considère une hypothèse supplémentaire, celle de dérivabilité de la fonction sur l'intervalle. La dernière méthode, celle de la sécante, permet quant à elle de palier à la non dérivabilité de la fonction réelle considérée. De plus, nous avons ainsi une évolution graduelle de la complexité des méthodes, autant au niveau de leur compréhension que de leur transposition en algorithme informatisé et en code Python. Plus de détails sur ces différents aspects seront donnés en temps voulu. Cet ordre est inchangé par rapport à la séquence source.

Méthode de dichotomie

Notre première méthode est celle de dichotomie, que les élèves ont déjà eu l'occasion de rencontrer et d'appliquer auparavant. Elle fait partie de nos prérequis et est basée sur le Théorème des valeurs intermédiaires. En guise d'introduction se trouve un rappel de ce théorème, une illustration graphique ainsi que la particularisation de ce dernier à la recherche de zéro d'une fonction. Ensuite viennent les explications concernant le fonctionnement de la méthode de dichotomie ainsi que les différents cas qui peuvent se présenter lors de son utilisation, présentés ci-dessous.

> L'idée de base de l'approche par dichotomie est simple. En effet, elle consiste à couper l'intervalle $]a, b[$ en deux en considérant le point milieu $m = \frac{(a+b)}{2}$.
>
> Dès lors, plusieurs cas sont possibles :
>
> 1. si $f(m) = 0$, alors nous avons trouvé une racine de f dans l'intervalle $]a, b[$,
> 2. si $f(a)$ et $f(m)$ sont non nuls et de signes opposés, alors la recherche continue dans l'intervalle $]a, m[$,
> 3. sinon, $f(m)$ et $f(b)$ non nuls et de signes opposés, alors la recherche continue dans l'intervalle $]m, b[$.

Une fois la méthode explicitée, vient le tour de l'écriture du premier algorithme. Avant d'effectuer l'analyse de la situation dans laquelle les élèves se trouvent face celui-ci, nous allons discuter de la raison pour laquelle nous avons intégré des algorithmes en pseudo-code dans notre séquence. De fait, il s'agit d'un ajout par rapport à la séquence source qui ne considère pas, en tout cas de façon explicite, le passage de la résolution mathématique du problème à l'algorithme informatisé avant l'implémentation en Python. Or, nous avons vus dans la section 4.4, dans notre cadre théorique des TICE, que la tâche qui consiste à écrire un programme afin de résoudre un problème comme le nôtre implique une double transposition. Cependant, nous avons aussi vu que le passage par l'algorithme en pseudo-code n'est pas indispensable pour qui maitrise la programmation. Or, ce n'est pas le cas de notre public qui, bien qu'il connaisse les structures liées à l'algorithmique, n'a pu les expérimenter qu'en Scratch, un langage qui lui n'induit pas le recours au pseudo-code sur

papier, contrairement à un langage de programmation expert comme Python. De plus, nous avons vu que ces différentes phases poussent les élèves à tenir compte des possibilités de l'outil, ainsi que de son fonctionnement, et ce afin que les algorithmes puissent être transposés en programmes qui « fonctionnent ». Autrement dit, ces étapes permettent de tenir compte de la transposition informatique, de l'instrumentation et de la pseudo-transparence. Enfin, en dehors du contexte des TICE, nous avons aussi discuté du fait que confronter les élèves à une même tâche dans deux environnements différents (informatisé et papier-crayon) permet d'atteindre une meilleure compréhension des objets manipulés, qui sont ici des algorithmes. Tous ces arguments justifient donc notre choix de travailler l'algorithmique et la programmation non seulement sur la base de programme mais aussi d'algorithmes en pseudo-code.

Algorithme

Le premier algorithme informatisé de notre séquence est celui associé à une itération de la méthode de dichotomie. Les élèves, de façon autonome et sans intervention de la part de l'enseignant, doivent écrire cet algorithme, ce qui correspond à l'un de nos objectifs d'apprentissage : « Décrire des algorithmes en pseudo-code ». Aucune indication n'est donnée concernant la manière d'écrire cet algorithme, de cette façon chaque élève peut s'orienter vers ce qui lui semble le plus adapté, faire son propre cheminement de pensée, se poser des questions et ce sans l'intervention de l'enseignant. Autrement dit, nous nous trouvons ici dans une situation a-didactique mettant l'élève en position de chercheur s'interrogeant sur la façon de décrire cette méthode sous la forme d'un algorithme, ce qui implique une dévolution.

Pour la résolution de cette tâche, les élèves doivent se référer à la structure de la méthode qui vient d'être énoncée, les différents cas qui peuvent se présenter, mais aussi à leurs connaissances préalables en matière d'algorithmique. En effet, ils connaissent les différentes structures pouvant composer un algorithme et peuvent, à partir des informations qu'ils ont sur le problème, se rendre compte qu'ils doivent utiliser ici une structure du type « si/alors/sinon », en l'adaptant au contexte. Dans les faits, la forme attendue est celle de l'algorithme 2.

Lors de la phase d'action de cette situation, deux stratégies de résolution de la tâche peuvent être déployées. D'un coté, il est fort probable que la majorité des élèves opte pour un langage naturel, se référant à la description de la méthode faite auparavant ou se remémorant les algorithmes qu'ils ont pu rencontrer en Scratch. Par exemple, en lieu et place de la condition $f(a) * f(m) < 0$, ces élèves noteraient « $f(a)$ et $f(m)$ sont non nuls et de signes opposés ». De l'autre, il n'est pas à exclure que certains élèves prennent les devants et se servent de la fonction Python suivant l'algorithme pour compléter ce dernier, passant ainsi du programme à l'algorithme informatisé et non l'inverse. Ce cas de figure n'est certes pas celui recherché mais il ne va pas non plus à l'encontre de nos objectifs d'apprentissage car, nous l'avons rappelé, écrire un programme pour résoudre un problème implique généralement de multiples allers-retours entre algorithme informatisé et programme.

Algorithme 2 : Une itération de la méthode de dichotomie

Entrées : f une fonction continue, a, b deux nombres encadrant une racine de f

1 m est le milieu du segment $]a, b[$
2 **si** $f(m) = 0$ **alors**
  Sorties : m
3 **sinon si** $f(a) * f(m) < 0$ **alors**
  Sorties : l'intervalle $]a, m[$
4 **sinon**
  Sorties : l'intervalle $]m, b[$
5 **fin**

Pour ce qui est de la phase de formulation, il est attendu des élèves qu'ils communiquent (oralement ou en se déplaçant au tableau) à l'enseignant leurs propositions d'écriture de l'algorithme. Une comparaison de ces soumissions a ensuite lieu, elle peut prendre la forme d'un débat incluant l'ensemble de la classe, mais nous ne considérons pas que nous impliquons les élèves dans un processus de validation, dans le sens où nous n'attendons pas d'eux qu'ils fournissent de preuve de leurs assertions.

Suite à cette formulation, nous retrouvons une phase d'institutionnalisation qui, portée par le professeur, met en évidence ce qui était attendu des élèves et ce qu'ils doivent en retenir. Cette institutionnalisation porte sur plusieurs points. Le premier concerne l'apparence de l'algorithme 2 et plus précisément le langage utilisé, nommé pseudo-code. Ce dernier est très largement exploité pour décrire des algorithmes, d'autant plus lorsqu'ils sont destinés à être exécuté par un ordinateur, et sera considéré dorénavant comme représentation usuel de ces derniers. Pour ce cours, nous nous sommes inspirés ici de la façon dont Simon Modeste décrit les algorithmes informatisés dans sa thèse [Modeste, 2012]. Le second point quant à lui concerne les éléments qui doivent être présents dans un algorithme en pseudo-code, c'est-à-dire les entrées, les sorties et les différentes instructions qui le composent, qui se trouvent ici être des instructions conditionnelles. La façon d'écrire de ces instructions conditionnelles constitue le dernier point de cette institutionnalisation.

Dans la situation que nous venons de décrire, nous pouvons considérer que la méthode à décrire en pseudo-code est une variable didactique. En effet, nous le verrons dans la suite, dans une autre version de la méthode ou pour une méthode différente, l'enchainement et le type d'instructions qui composent l'algorithme peuvent être différents, ainsi même que les entrées et sorties. Par conséquent, la production des élèves et les connaissances développées lors de la tâche en sont affectées.

Fonction en Python

Directement après l'algorithme, nous retrouvons la fonction Python de la figure 5.1. Les élèves disposent du code complet de cette dernière ainsi que des arguments d'entrée

```
def dichotomie_iter(f,a,b):
    m = (a+b)/2
    if f(m) == 0 :
        a = m
        b = m
        return a, b
    if f(a)*f(m) < 0 :
        b = m
        return a, b
    else :
        a = m
        return a, b
```

Figure 5.1 – Code Python de la fonction `dichotomie_iter`.

et sont supposés, au travers des deux questions associées, reconnaitre dans ce code l'algorithme qu'ils viennent d'écrire. La première question dirige les élèves dans leur analyse, en leur indiquant d'identifier les sorties la fonction, tandis que la seconde leur demande d'en tirer une conclusion. Autrement dit, il faut ici reconnaitre les similarités mais aussi les différences entre les deux représentations de l'algorithme, celle en pseudo-code et celle en Python. Pour cette situation, le professeur dialogue avec les élèves en vue de dégager les différentes informations importantes.

Nous l'aurons compris, ces questions traitent en réalité de la transposition subie par l'algorithme lors de sa transformation en programme. Plus précisément, il s'agit de la deuxième phase de la double transposition de Briant dont nous avons discuté dans le chapitre 4. Cela signifie que, à l'issue des interactions avec la classe, l'enseignant doit effectuer une synthèse des différences entre l'algorithme en pseudo-code et le programme. La première différence, et la plus visible, concerne la syntaxe en Python. Concrètement, elle se traduit par la distance entre la structure de l'algorithme et celle de la fonction en Python, par exemple en ce qui concerne la séquence d'instructions conditionnelles « if/then/else ». Rappelons d'ailleurs que les élèves ont accès aux conventions d'écriture des instructions en Python via leur aide-mémoire. La seconde concerne quant à elle la transformation des expressions telles que *m est le milieu du segment* $]a, b[$ ou encore la condition $f(m) = 0$ qui deviennent respectivement l'affectation de variable $m = (a+b)/2$ et `f(m) == 0`. Nous retrouvons là des éléments liés aux phénomènes de pseudo-transparence entre objets mathématiques et objets informatiques, d'instrumentation et de transposition informatique. Reconnaitre ces différences et distinguer un algorithme informatisé d'un programme peut être vu comme une extension des objectifs d'apprentissage « décrire un algorithme en pseudo-code » et « réaliser un algorithme avec un logiciel adapté ».

Exercices

Une fois que la fonction `dichotomie_iter` a été ajoutée par les élèves à leur notebook, l'étape suivante consiste à tester son exécution sur un exemple concret, la fonction sinus,

pour laquelle nous considérons l'intervalle $]3, 4[$ encadrant la racine $x = \pi$. Les élèves doivent donc faire appel à la fonction `dichotomie_iter` en lui transmettant comme arguments `fsin`, 3 et 4. Il leur est demandé de déterminer le milieu de l'intervalle que renvoie cette fonction, valant 3.25, et de juger intuitivement de sa qualité en tant qu'approximation de *pi*. Étant donné qu'il s'agit d'un résultat qui semble insatisfaisant, il est demandé de réfléchir à une nouvelle stratégie qui permettrait d'obtenir une meilleure approximation de la racine recherchée.

Dans un premier temps, les élèves travaillent seuls sur leur ordinateur tandis que le professeur se tient à disposition pour toute question ou difficulté concernant l'appel de fonction, la notion de programmation étant réactivée ici. Pour ce qui est de l'interprétation des résultats, deux points de vue peuvent émerger : certains élèves peuvent considérer ce résultat comme une bonne approximation de π, tandis que d'autres non. Dans les faits, la majorité des élèves devraient se rendre compte qu'il y a une erreur trop importante par rapport à la vraie valeur de la racine. Si ce n'est pas le cas, il incombe au professeur de le signaler.

Le fait que l'approximation obtenue soit peu satisfaisante permet comme nous l'avons vu d'amener la réflexion autour d'une stratégie nouvelle, plus performante. Plusieurs propositions d'élèves sont possibles, comme par exemple le fait de considérer un intervalle encore plus restreint ou encore l'utilisation d'une toute autre méthode que celle de dichotomie, mais nous cherchons ici à faire émerger au moins une stratégie bien précise. En effet, la solution la plus évidente est d'effectuer plusieurs itérations de la méthode de dichotomie tandis qu'une autre serait d'imposer la longueur de l'intervalle d'arrivée, autrement dit d'imposer une certaine précision à l'arrivée. Il est probable que les élèves arrivent seuls à la conclusion qu'effectuer une boucle, donc plusieurs itérations, permettrait d'atteindre une meilleure approximation, mais nous ne nous attendons pas à ce qu'ils aient l'idée d'imposer une précision.

Avant de passer à la prochaine étape de notre séquence de cours, nous allons nous attarder sur la vision de l'algorithme véhiculée jusqu'à présent. Tout d'abord, dans la section « Mise en place », les méthodes algorithmiques abordées dans le cours ont été présentées comme des moyens de résolution de notre problème de recherche de zéro de fonction. Cela correspond à l'aspect problème de l'algorithme que nous avons développé au chapitre 1. Ensuite, le premier algorithme de la séquence a été décrit en pseudo-code. Or, de par sa nature, l'écriture en pseudo-code induit la non-ambiguïté des étapes de l'algorithme. De outre, cet algorithme fonctionne sur un nombre fini de données et est destiné à être exécuté par un ordinateur sous la forme d'un programme. L'ensemble de ces éléments font référence quant à eux référence à l'aspect effectivité de l'algorithme. En résumé, les deux aspects problème et effectivité sont présents, ce qui signifie que l'algorithme est considéré ici en temps qu'outil au service des mathématiques.

Version itérative

Étant donné que la première approche ne permet pas d'obtenir une approximation satisfaisante du zéro recherché de la fonction sinus, nous adoptons la nouvelle stratégie qui

consiste, comme nous venons de le voir, à effectuer une boucle afin d'obtenir un meilleur encadrement de la racine. Dans cette séquence, nous appelons cette démarche la « version itérative » de la méthode de dichotomie (voir algorithme 3).

Ici encore, il est demandé aux élèves d'écrire un algorithme en pseudo-code. Ce dernier prend en entrée une fonction f, les nombres a et b encadrant une racine de cette fonction ainsi que le nombre n d'itérations de la boucle et renvoie l'approximation de la racine recherchée dans $]a, b[$ obtenue après les n itérations. Contrairement à la situation a-didactique précédente, les élèves possèdent cette fois plus d'information sur la façon dont ils doivent écrire l'algorithme. Cependant, ce dernier n'est plus composé des mêmes instructions que l'algorithme 2 puisqu'il est question ici d'une structure itérative du type boucle `for`.

Pour réaliser cette tâche, les élèves doivent réinvestir les connaissances qu'ils ont acquises en ce qui concerne les algorithmes en pseudo-code et leur structure ainsi que celles que nous considérons comme prérequis pour ce cours, plus particulièrement la notion de structure itérative avec un nombre d'itérations donné. La connaissance visée ici est la manière d'écrire une boucle de ce type en pseudo-code. En effet, en Scratch ce type d'instruction se traduit par une instruction du type « répéter n fois (instructions) » tandis qu'ici une forme attendue serait par exemple « pour $i = 0$ à $n - 1$ faire (instructions) ». Les élèves devraient être capables d'écrire, au moins partiellement, l'algorithme sans trop d'interventions guidées de la part du professeur. Ce dernier peut construire l'algorithme au tableau en fonction des propositions de la classe ou les laisser travailler individuellement avant de faire une correction.

Outre les multiples possibilités de syntaxe pour la boucle `for`, les élèves sont susceptibles d'écrire leur algorithme de deux façons différentes. La première, celle attendue, fait appel à la fonction `dichotomie_iter` vue précédemment au sein de la boucle. La seconde consiste plutôt à réécrire au sein de cette boucle l'entièreté de la structure conditionnelle de l'algorithme précédent. Ces deux options sont tout à fait viables, cependant la première est plus adaptée. En effet, la décomposition d'un problème en sous-problèmes est un aspect très présent en programmation, nous pouvons d'ailleurs voir cela comme une compétence transversale à travailler. Par conséquent, tout comme l'IREM, nous privilégions l'appel à la fonction `dichotomie_iter` plutôt que sa réécriture au sein de la boucle.

Algorithme 3 : Méthode de dichotomie - version itérative

Entrées : f une fonction continue, a et b deux nombres, n un entier.

1 **pour** $i = 0$ **à** $n - 1$ **faire**

2 | Appliquer la fonction `dichotomie_iter` sur l'intervalle $]a, b[$.

3 **fin**

4 Calculer le milieu m du dernier intervalle ainsi déterminé.

Sorties : m

Une fois l'algorithme corrigé, nous entrons dans une nouvelle situation a-didactique, poursuivant un objectif différent de la précédente. En effet, pour la première fois il est demandé aux élèves d'implémenter par eux-même un algorithme en Python (voir figure 5.2). Bien que plusieurs codes Python aient été abordés au préalable, les élèves n'ont encore jamais eu l'occasion de s'essayer à l'écriture d'un code. Deux apprentissages sont visés ici : « écrire un programme avec des instructions élémentaires » et « programmer un calcul itératif, le nombre de répétition étant donné ».

Durant la phase d'action, les élèves sont en interaction avec le logiciel d'édition de code et, avec l'aide de l'algorithme informatisé et de l'aide-mémoire, s'essayent à la programmation en Python. Comme nous l'avons vu, le milieu informatisé auquel les élèves sont confrontés – le notebook Jupyter – présente une rétro-action, c'est-à-dire à qu'il réagit à leurs tentatives d'exécution de code, qu'elles soient fructueuses ou non. Ainsi, les élèves adaptent leurs actions en fonction des informations communiquées par l'éditeur de code au travers de son interface mais aussi du fonctionnement même de ce logiciel. Nous nous trouvons d'ailleurs dans la deuxième étape de la double transposition, celle de la transformation de l'algorithme informatisé en programme, ce qui prend en compte l'instrumentation, la transposition informatique des notions mathématiques dans l'environnement informatisé et la pseudo-transparence entre les objets et concepts.

Pour ce qui est de la phase de formulation, nous considérons ici qu'elle se fait via la communication avec un pair. Les élèves travaillent par groupe de deux et communiquent sur base des résultats qu'ils ont obtenu, comparent leur production respective et s'aident à mutuellement si nécessaire. La référence à un autre membre du triangle didactique que l'enseignant peut être interprété au regard du contrat didactique. En effet, jusqu'à présent la formulation mettait en scène l'ensemble de la classe en interaction avec l'enseignant, la nouvelle approche proposée ici constitue donc une rupture du contrat.

Viens ensuite la phase de validation. Lors de cette dernière, il est attendu que les élèves se confrontent, par groupes de deux établis lors de la phase précédente, et qu'ils débattent, si nécessaire, sur les codes qu'ils ont obtenus après la recherche personnelle et la mise en commun avec leur voisin. Pour appuyer leurs propositions, ils doivent se référer aux différents codes en Python qui ont été traités jusque là dans la séquence, ainsi qu'à l'aide-mémoire.

Enfin, la situation a-didactique se clôture sur une institutionnalisation. Le professeur présente à la classe la version attendue de la fonction `dichotomie_nbpas` (voir figure 5.2) et insiste sur les apprentissages visés, notamment la forme que prend une instruction conditionnelle de ce type dans un programme en Python. Il convient par ailleurs que le professeur rappelle aux élèves qu'ils ont déjà rencontré au sein de cette séquence la fonction `range` et son fonctionnement.

Nous avons donc ici une séquence complète action-formulation-validation suivie d'une institutionnalisation comme le préconise Brousseau. Rappelons cependant qu'une situation a-didactique ne comportant pas de phase de validation n'est pas pour autant inintéressante. En effet, nous avons vu dans notre section sur la théorie des situations didactiques

```
def dichotomie_nbpas(f,a,b,n):
    for i in range(n) :
        a, b = dichotomie_iter(f,a,b)
    m = (a+b)/2
    return m
```

Figure 5.2 – Code Python de la fonction `dichotomie_nbpas`.

que la succession des quatre étapes ne constitue pas une règle à suivre à tout prix et que d'autres approches peuvent tout à fait aboutir à l'apprentissage visé.

L'étape suivante consiste à tester cette fonction. Pour ce faire il est demandé aux élèves de faire appel à la fonction en lui communiquant comme argument la fonction `fsin`, l'intervalle $]3, 4[$ et un nombre de pas égal à 10. L'intérêt de cet exercice réside dans le fait qu'il permet de rendre compte de l'efficacité de cet algorithme en comparaison avec le précédent. En effet, la racine obtenue grâce à cette version itérative est d'environ 3.1411, ce qui est plus proche de la racine réelle π que l'approximation précédente.

Enfin, dans cette partie consacrée à la version itérative de la méthode de dichotomie, ce sont à nouveau les aspects problème et effectivité de l'algorithme qui transparaissent. L'algorithme est un outil permettant d'approcher la racine de la fonction sinus recherchée.

Version conditionnelle

À l'instar de l'activité de l'IREM de Paris [IREM de Paris, 2018], nous proposons dans ce cours une deuxième version de chacune de nos méthodes : une version conditionnelle. L'introduction de cette deuxième stratégie a deux raisons d'être. La première est qu'elle permet d'obtenir une approximation encore plus satisfaisante que celle de la version itérative. La seconde est que la version conditionnelle nécessite, comme son nom l'indique, l'introduction de la structure de boucle conditionnelle `while`, à la fois dans les algorithmes et dans les programmes, ce qui nous permet de travailler nos objectifs d'apprentissage.

L'approche pour l'algorithme de la version conditionnelle de la méthode de dichotomie est la même que celle choisie pour la version itérative. Nous n'allons donc pas la parcourir en détails comme nous avons pu le faire alors. Nous nous contenterons d'indiquer que les élèves sont mis dans une situation de réflexion ayant pour but d'écrire l'algorithme qui prend en entrée la fonction f, les nombres a et b encadrant un zéro de f, $e > 0$ la précision souhaitée et qui renvoie une valeur approchée d'une racine de f dans l'intervalle $]a, b[$, à e près.

Il nous faut cependant discuter des spécificités de cette nouvelle situation, bien qu'elle suive la même progression. La connaissance visée ici est celle de l'écriture en pseudo-code d'une instruction conditionnelle, c'est à dire un calcul itératif avec une fin de boucle conditionnelle, et les élèves disposent d'assez de connaissances pour effectuer cette tâche. En effet, ils devraient maitriser à présent la structure d'un algorithme en pseudo-code

et connaissent la notion de boucle conditionnelle puisqu'il s'agit d'un de nos prérequis. En Scratch, ce type d'instruction se présente notamment sous la forme « répéter jusqu'à (condition) (instructions) ».

Il subsiste néanmoins deux difficultés dans cette situation, la première étant l'appréhension de la notion de « précision ». En effet, la traduction de « à e près » n'est pas forcément intuitive. L'intérêt de cette étape est donc de laisser l'opportunité aux élèves, sur papier, de rechercher, tester et se tromper avant de penser à leur communiquer la correction. Lors de cette dernière, la transposition informatique qui a lieu entre cette notion de précision et sa représentation en pseudo-code doit être évoquée. Cette dernière se manifeste au niveau de la condition de boucle « distance(a,b) $>$ e », comme nous pouvons le voir sur l'algorithme 4. La seconde difficulté réside dans le fait que les boucles conditionnelles nécessitent une anticipation de la condition de fin de la boucle. Raison pour laquelle nous avons vu que Briant les considère comme moins accessibles que les autres boucles. Ces deux obstacles sont donc liés.

Vient ensuite la phase d'implémentation de l'algorithme en Python. Dans cette situation, le professeur n'intervient pas comme possesseur du savoir et laisse à nouveau les élèves tirer parti de la phase d'action pour s'exercer à l'écriture d'un programme comportant un calcul itératif avec une fin de boucle conditionnelle. En plus de l'algorithme informatisé et de la rétro-action du logiciel d'édition de code pour les guider, les élèves doivent se référer à leur aide-mémoire pour prendre connaissance de la syntaxe des boucles `while` en Python. La fonction `dichotomie_eps` peut revêtir plusieurs formes, mais celle globalement attendue est représentée à la figure 5.3.

La phase d'action doit être suivie d'une phase de formulation, mettant cette fois-ci en communication le professeur et l'ensemble de la classe, et d'une phase d'institutionnalisation. Cette dernière se porte sur la structure de la boucle conditionnelle et doit mettre en évidence la transposition qui a eu lieu entre la *distance(a,b)* dans l'algorithme informatisé, et son homologue en Python, `abs(b-a)`. Notons que `abs` est une fonction prédéfinie à laquelle nous avons accès grâce au module `math` et qu'il s'agit de l'équivalent en Python de la valeur absolue en mathématiques.

Pour cette version conditionnelle, nous avons à nouveau quelques exercices, deux plus précisément. Ces derniers sont différents des précédents dans le sens où ils introduisent

Algorithme 4 : Méthode de dichotomie - version conditionnelle

Entrées : f une fonction continue, a et b deux nombres, $e > 0$

1 **tant que** *distance*$(a, b) > e$ **faire**

2 | Appliquer la fonction `dichotomie_iter` sur a et b

3 **fin**

4 Calculer le milieu m du premier intervalle $]a, b[$ tel que distance$(a, b) = e$

Sorties : m

```
def dichotomie_eps(f,a,b,e):
    # n = 0
    while abs(b - a)>e :
        a, b = dichotomie_iter(f,a,b)
        # n += 1 ou n = n+1
    m = (a+b)/2
    # print(n)
    return m
```

Figure 5.3 – Code Python de la fonction `dichotomie_eps`.

une nouvelle dimension de l'algorithmique, celle de la comparaison entre les solutions algorithmique et de leur optimalité.

Le premier exercice est subdivisé en deux sous-points. Tout d'abord, il est demandé aux élèves d'appliquer la version conditionnelle de la méthode de dichotomie sur la fonction sinus, toujours sur l'intervalle $]3, 4[$ et avec une précision donnée de 10^{-5}, le résultat attendu étant 3.1416 une fois arrondi. Il s'agit d'une approximation de π que l'on peut considérer comme satisfaisante à ce stade. Des erreurs d'exécution peuvent subvenir lors de l'appel de la fonction. En effet, l'expression 10^{-5} en papier-crayon se traduit par $10**(-5)$ en Python, il s'agit d'un manque de transparence entre les deux environnements. Cette information est par ailleurs disponible dans l'aide-mémoire.

Ensuite, les élèves se retrouvent face à une tâche nouvelle, qu'ils ont cependant probablement déjà rencontrée en Scratch, celle de déterminer le nombre d'itération de la boucle lors d'une exécution du programme. En d'autres mots, il leur est demandé de déterminer le nombre d'itérations nécessaires pour atteindre la précision demandée. Ce type de manipulation demande d'ajouter un compteur n à la fonction qui vient d'être définie. Cette variable doit être affectée à la valeur 0 avant la boucle et être incrémenté de 1 à l'intérieur de celle-ci. Ainsi, en fin d'exécution, la valeur associée au compteur n correspondra au nombre d'exécutions de la boucle avant que la condition de sortie de boucle ne soit atteinte. Les lignes de code qui correspondent à ces manipulations sont insérées en commentaire dans la fonction `dichotomie_eps` de la figure 5.3. Pour récupérer le nombre d'itérations, les élèves peuvent soit avoir recours à la fonction prédéfinie `print`, qu'ils peuvent retrouver dans leur aide-mémoire, soit considérer la variable n comme une sortie de la fonction. Néanmoins, il est préférable que les élèves optent pour la première solution et que le professeur insiste sur la distinction à faire entre affichage à l'écran et sortie d'un algorithme, réponse de l'algorithme à la question du problème – et ce afin d'éviter tout amalgame entre les deux notions. Par ailleurs, le professeur peut indiquer aux élèves qu'il existe deux notations possibles pour l'indentation de la variable n : `n = n+1` et `n +=1`. Ces notations d'affectation de variable informatique peuvent poser quelques problèmes causés d'un côté par l'utilisation du symbole de l'égalité et de l'autre par la syntaxe spécifique `+=`. Nous avons discuté de ces phénomènes dans notre cadre théorique des TICE. Enfin, la réponse attendue à cette question est que la méthode de dichotomie, dans sa version itérative, effectue 17 itérations avant d'atteindre la convergence en 3.1416 pour une précision de 10^{-5}.

Le second exercice va plus loin dans l'analyse des résultats et des performances de l'algorithme. Il y est question de répertorier au sein d'un tableau le nombre d'itérations nécessaires à la fonction `dichotomie_eps` pour approximer la racine de la fonction sinus comprise entre les abscisses 3 et 4 pour précision variant de 10^{-3} à 10^{-10}. Il s'agit là d'une modification intentionnelle de la variable didactique que représente la précision e et qui, nous allons le voir, induit un saut informationnel en faisant apparaitre les limites de la stratégie de résolution que nous utilisons actuellement. En effet, l'intérêt de cet exercice est de mettre en évidence l'évolution du nombre d'itérations en fonction de la précision demandée, c'est-à-dire son augmentation lorsque la précision augmente.

Ces exercices ont deux particularités. La première est qu'ils intègrent la notion de vitesse de convergence par le biais du nombre d'itérations nécessaires à l'algorithme avant d'atteindre la précision e souhaitée. La seconde est qu'ils permettent d'introduire auprès des élèves la question du compromis entre vitesse de convergence et précision. En effet, dans le tableau les résultats varient de 10 à 34 itérations et le constat est sans appel :

> imposer une précision relativement importante a pour conséquence d'augmenter significativement le nombre d'itérations nécessaires avant d'atteindre la convergence.

Ces questions autour de la vitesse de convergence mènent par extension au concept d'optimalité d'un algorithme. En s'intéressant à la vitesse de convergence des différents algorithmes composant notre séquence, nous allons pouvoir les comparer entre eux et en tirer des conclusions. Dans notre cadre théorique de l'algorithmique, nous avons vu que le critère le plus utilisé pour comparer des solutions algorithmiques est la complexité, or ici elle n'est pas abordée et ne fait pas partie de nos objectifs d'apprentissage. Par contre, nous regroupons ces questionnements dans l'objectif intitulé « s'interroger sur l'efficacité d'un algorithme ».

Enfin, nous retrouvons à nouveau la dimension outil de l'algorithme car nous nous intéressons à sa capacité à résoudre notre problème de recherche d'un zéro d'une fonction en un nombre fini d'étapes non-ambiguës (caractéristique induite par l'écriture en pseudo-code), agissant sur des données en nombre fini, proposant une solution à notre problème en un temps fini (le nombre d'itérations nécessaire en étant une indication) pour l'instance considérée et destiné à termes à être implémenté sous la forme d'un programme informatique. Il s'agit là de la vision de l'algorithme que nous avions déjà dégagées auparavant.

Méthode de Newton

Notre nouvelle méthode est appelée la méthode de Newton. Par rapport à la méthode de dichotomie, celle-ci considère une famille d'instances légèrement différente, celle des fonctions continues et dérivables. Cela signifie que pour appliquer cette méthode nous devons avoir accès à la dérivée de la fonction considérée.

Plus nous avançons dans le cours, plus les sections sont courtes. En effet, à ce stade, les élèves ont été en contact avec l'ensemble des structures algorithmiques qui nous intéressent

ainsi qu'à la majorité des objectifs d'apprentissages. Par conséquent, nous considérons que les algorithmes et les fonctions en Python peuvent être abordés de manière plus frontale, à l'image de ce que nous pouvons retrouver dans la séquence source [IREM de Paris, 2018].

La méthode de Newton, contrairement à celle de dichotomie, ne fait pas partie de nos prérequis. Dès lors, et comme nous ne visons pas ici d'apprentissage mathématique, nous nous calquons sur l'approche choisie par l'IREM et nous présentons aux élèves uniquement les informations sur la méthode dont ils ont besoin pour travailler correctement sur nos objectifs.

La méthode de Newton est itérative, cela signifie que, du point de départ x_0 – suffisamment proche de la racine recherchée –, chaque nouveau point x_{n+1} est calculé à partir de du point précédent, x_n. La mise à jour de la suite $(x_0, x_1, \ldots, x_n, x_{n+1})$ se fait comme suit.

> Le point x_{n+1} est défini comme l'abscisse du point d'intersection entre la tangente T à la courbe de f au point d'abscisse x_n et l'axe des abscisses.

Afin d'aider à la compréhension de la méthode, une illustration de la méthode est proposée, représentant sa progression d'un point x_0 vers un point x_3 proche de la racine de la fonction f. Ensuite, le terme général de notre suite et la condition de la convergence de la suite sont exposés. Pour les raisons évoquées plus haut, le terme général de la suite est donné et le processus de détermination de celui-ci n'est pas plus développé que nécessaire. L'expression de ce dernier est

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

et la suite (x_n) converge vers la racine recherchée, la solution à notre problème, à condition que la dérivée $f'(x_n)$ soit non nulle.

Avant d'aborder la version itérative de la méthode de Newton, il est plus intéressant de définir dans un premier temps la fonction Python permettant d'effectuer une itération de notre méthode (voir figure 5.4). Comme dans la séquence source, la fonction `newton_iter` est donnée aux élèves. Cette dernière prend comme arguments une fonction, sa fonction dérivée ainsi qu'un point et renvoie le point obtenu en appliquant une itération de la méthode de Newton telle qu'elle vient d'être définie. Cet apport direct est lié aux objectifs d'apprentissage concernant les instructions élémentaires, c'est-à-dire l'écriture en instructions élémentaires d'une formule permettant un calcul et l'écriture d'un programme calculant et donnant la valeur d'une fonction. Nous justifions la fluctuation entre apport direct et construction des savoirs par les élèves en nous reposant sur Régine Douady qui

```
1 def newton_iter(f,df,x0) :
2         xn = x0 - f(x0)/df(x0)
3         return xn
```

Est ce que les élèves doivent écrire l'expression de la dérivée et le calcul de f'en x = x0 sur papier ou sur le logiciel ?

Figure 5.4 – Code Python de la fonction `newton_iter`.

explique dans sa thèse [Douady, 1984] que certaines notions peuvent tout à fait être apportées par la lecture d'un manuel ou par apport direct du professeur par exemple. Notons que Douady se base elle-même sur la théorie des situations didactiques de Brousseau et en élargit les considérations. Aussi, afin de faciliter le suivi de la progression des élèves ainsi que la correction, il est plus simple à ce stade que l'ensemble de la classe adopte les mêmes noms de variables, x_0 et x_n.

Version itérative

Puisque nous savons qu'il n'est pas intéressant de s'attarder au résultat associé à une seule itération de la méthode, à cause du manque de précision de l'approximation obtenue, la séquence poursuit directement par la version itérative. L'algorithme informatisé 5 est donné aux élèves sous sa forme complète, ce qui constitue en soi une rupture du contrat didactique puisqu'il s'agit du premier algorithme en pseudo-code que les élèves n'ont pas à écrire par eux-même.

L'objectif poursuivit ici est l'interprétation d'un algorithme plus complexe. En effet, l'algorithme 5 présente une difficulté supplémentaire par rapport à tous ceux rencontrés jusque ici, il contient une instruction de « mise à jour » de la variable informatique x_0. Il s'agit là d'une opération intrinsèque à la méthode de Newton ainsi qu'à la suivante, celle de la sécante. L'enseignant peut parcourir l'algorithme avec la classe et interpréter cette instruction avant la transposition en programme, c'est-à-dire l'identifier comme l'affectation à la variable x_0 de la valeur de la variable x_n calculée à la ligne précédente, ou alors laisser les élèves chercher par eux-même pendant la résolution de la tâche suivante, consistant à implémenter cet algorithme informatisé en Python. La fonction `newton_nbpas` (voir figure 5.5) doit effectuer un nombre n d'itérations de la méthode de Newton sur une fonction f à partir du point de départ x_0 et renvoyer le point x_n ainsi obtenu, c'est-à-dire la solution algorithmique de la méthode de Newton dans sa version itérative.

Cet implémentation peut être interprétée comme un exercice de familiarisation et de réinvestissement des connaissances que les élèves ont acquis jusqu'à présent. En effet, ces derniers sont tout à fait à même d'écrire la fonction dans son intégralité puisque les notions d'instructions itératives ont déjà été introduites avec la méthode de dichotomie, que ce soit sous la forme de pseudo-code ou sous la forme de programme en Python. Encore une

Algorithme 5 : Méthode de Newton - version itérative

Entrées :
f une fonction continue et dérivable, *df* sa dérivée, x_0 un réel et n un entier positif

1 **pour** $i = 0$ **à** $n - 1$ **faire**
2 | Appliquer la fonction `newton_iter` en x_0 pour déterminer le point x_n.
3 | Mise à jour de x_0
4 **fin**

Sorties : x_n

```
def newton_nbpas(f,df,x0,n):
    for i in range(n) :
        xn = newton_iter(f,df,x0)
        x0 = xn
    return xn
```

FIGURE 5.5 – Code Python de la fonction `newton_nbpas`.

fois, la difficulté réside dans la transposition entre algorithme informatisé et programme au niveau de l'instruction de mise à jour. Dans cette situation, les élèves peuvent se référer au professeur en cas de problème.

Version conditionnelle

Pour la deuxième version de la méthode de Newton, la première situation rencontrée est à nouveau un exercice d'écriture. Contrairement à ce qui a été fait pour la méthode de dichotomie, et comme c'est le cas dans la séquence de l'IREM, ici la condition de la boucle est donnée afin de ne pas rajouter de difficulté supplémentaire et de se concentrer sur l'apprentissage visé, à savoir l'écriture d'un algorithme en pseudo-code et du programme correspondant. En effet, la structure de l'algorithme 6 est légèrement particulière : il n'y a pas un seul appel à la fonction `newton_iter` mais bien deux, un avant la boucle et un à l'intérieur de celle-ci.

Globalement, l'algorithme 6 est plus complexe que les précédents. Par conséquent, nous considérons que son écriture se fait avec l'aide du professeur, au tableau. Ainsi, les élèves sont amenés à réinvestir leurs connaissances, institutionnalisées précédemment, à les mettre en œuvre dans cet exercice d'écriture. Au fil des échanges entre l'enseignant et la classe, les élèves devraient percevoir que la boucle ayant comme condition $|x_n - x_0| > e$ ne peut être exécutée que si x_n est différent de x_0. En effet, sachant que la précision e est par définition supérieure à zéro, nous avons que

$$|x_n - x_0| > e > 0.$$

Algorithme 6 : Méthode de Newton - version conditionnelle

Entrées : f une fonction continue et dérivable, df sa dérivée, x_0 un réel et $e > 0$

Appliquer la fonction `newton_iter` en x_0 pour déterminer le point x_n

tant que $|x_n - x_0| > e$ **faire**

  Mettre à jour x_0

  Appliquer la fonction `newton_iter` en x_0 pour déterminer le point x_n

fin

Sorties : x_n

Or, si le point suivant x_n n'est pas déterminé avant la première exécution de la boucle, autrement dit si $x_n = x_0$, nous obtenons

$$|x_n - x_0| = |x_0 - x_0| = 0.$$

La condition à vérifier n'étant alors pas remplie, l'algorithme s'arrête.

L'écriture de la fonction `newton_eps` (voir figure 5.6) associée à l'algorithme 6 est par contre laissée à la charge des élèves, ce qui constitue une nouvelle fluctuation du contrat didactique. L'avantage d'avoir complété l'algorithme collectivement avant de demander l'implémentation en Python est qu'ils auront plus de facilité à corriger leurs potentielles erreurs si le logiciel d'édition de code leur indique un problème d'exécution. Durant la correction, il est important de mettre en évidence que lors de la transposition de l'algorithme informatisé vers la fonction en Python, l'expression $|x_n - x_0| > e$ devient `abs(xn-x0)` et la mise à jour de x_0 devient `x0 = xn`, comme pour la fonction `newton_nbpas`.

Exercices

Les deux exercices proposés ensuite suivent le même schéma qu'à l'accoutumée et sont tirés de la séquence de l'IREM [IREM de Paris, 2018]. Le premier consiste à tester la fonction Python `newton_nbpas` sur la fonction sinus, pour un nombre d'itération égal à 10 et à partir du point 4. Le résultat attendu est exactement de 3.141592653589793.

Le second quant à lui concerne la fonction Python `newton_eps`. Toujours à partir du point de départ $x0$ égal à 4, les élèves doivent tester la fonction associée à la version conditionnelle de la méthode pour une précision de 10^{-10} et en quantifier la vitesse de convergence, en nombre d'itérations, afin de la comparer à celle de la version conditionnelle de la méthode de dichotomie. Deux conclusions sont attendues. La première est que les deux versions de la méthode de Newton donnent le meilleur résultat jusqu'à présent, c'est à dire l'approximation la plus proche de la vraie valeur de π. Bien entendu, il s'agit toujours d'une approximation puisque nous sommes limités par le nombre de décimales qu'affiche à l'écran notre éditeur de code, c'est une contrainte due au dispositif utilisé. Il est cependant possible d'imposer une précision plus grande et de ce fait d'afficher un nombre plus important de décimales, mais ce n'est pas nécessaire ici. La deuxième conclusion est

```
1 def newton_eps(f,df,x0,e):
2     xn = newton_iter(f,df,x0)
3     # n = 0
4     while abs(xn-x0) > e :
5         x0 = xn
6         xn = newton_iter(f,df,x0)
7         # n += 1 ou n = n+1
8     # print(n)
9     return xn
```

FIGURE 5.6 – Code Python de la fonction `newton_eps`.

que la méthode de Newton, dans sa version conditionnelle, est beaucoup plus rapide que celle de dichotomie puisqu'elle converge, pour une même précision de 10^{-10}, en 4 itérations contre 34.

La difficulté dissimulée de ces questions réside dans l'accès à la dérivée de la fonction sinus, mais plusieurs stratégies sont possibles. D'un côté, il est possible d'écrire, en amont, une fonction `fcos` de la même forme que `fsin` afin d'y faire appel dans les arguments de `newton_eps`. De l'autre, et il s'agit de la solution la plus simple, il suffit de faire appel à la fonction Python `cos` proposée par le module `math` qui a été importé au préalable. Les codes correctifs de ces deux questions sont présentés à la figure 5.7. Observer le choix de stratégie des élèves peut s'avérer intéressant car il relève du processus d'instrumentation qu'ils ont engagé.

Méthode de la sécante

Notre dernière méthode est celle de la sécante. Très similaire à celle de Newton, elle s'en distingue néanmoins sur plusieurs points. Tout d'abord, il n'est pas nécessaire que la fonction considérée soit dérivable pour que la méthode de la sécante puisse s'appliquer. Ensuite, elle n'est pas construite à partir de la tangente à la courbe de la fonction, mais sur base d'une droite appelée ici la « sécante ». Enfin, elle démarre à partir de deux points x_0 et x_1 au lieu d'un seul.

Le principe de la méthode de Newton et celle de la sécante étant très proches, il n'est pas nécessaire que le professeur lors de son exposé y passe plus de temps que nécessaire. En outre, rappelons-le, nous ne cherchons pas ici à développer des compétences mathématiques mais bien des compétences en matière d'algorithmique et de programmation. Dans le support de cours, les différences que nous venons d'évoquer sont citées et les caractéristiques de la méthode de la sécante sont données point par point. Ces dernières sont résumées ci-dessous.

- Les points de départ x_0 et x_1 doivent être proches de la racine de f recherchée.

```
def fcos(x) :
    y = cos(x)
    return y

# version itérative
newton_nbpas(fsin,fcos,4,10)
# ou newton_nbpas(fsin,cos,4,10)

#version conditionnelle
newton_eps(fsin,fcos,4,10**(-10))
# ou newton_eps(fsin,cos,4,10**(-10))
```

Figure 5.7 – Codes Python des exercices de la méthode de Newton.

- À chaque itération, le point x_{n+1} est calculé à partir des points x_n et x_{n-1}.
- Le point x_{n+1} est l'abscisse du point d'intersection de la droite qui coupe la courbe de f aux points d'abscisses x_n et x_{n-1} [la « sécante »] et l'axe des abscisses.

Le terme général de la suite

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

est donné et non démontré. Pour finir, une illustration graphique de la méthode est proposée, comme pour les précédentes méthodes.

Exercices

Le reste de la partie sur la méthode de la sécante est présentée sous la forme d'exercices et les élèves doivent les aborder seuls. En effet, l'accompagnement dans les différents processus d'écriture des algorithmes informatisés et des codes en Python s'arrête ici. Afin de répondre aux différentes questions de ces exercices, les élèves doivent avoir recours non seulement à leur aide-mémoire, mais aussi à toutes les fonctions Python qui ont pu être implémentées précédemment. Autrement dit, ils sont faces à une situation ne faisant appel qu'à du connu et dans laquelle ils doivent réinvestir l'ensemble des connaissances qu'ils ont accumulées. Cela leur permet de tester leur compréhension, de juger de la validité de leurs connaissances et de travailler ce qu'on pourrait appeler leur « intuition algorithmique ». Notons que, bien qu'il n'en soit pas fait mention, il est attendu implicitement que les élèves passent par les deux phases de la double transposition, d'autant plus qu'il parait difficile que ces derniers puissent se passer de l'étape « papier-crayon » de l'algorithme informatisé avant l'implémentation des fonctions demandées en Python.

En réalité, nous conservons ici la posture adoptée par le groupe de recherche de l'IREM de Paris pour cette même méthode, avec comme seule différence l'ajout de questions relatives à la vitesse de convergence. Ces questions permettent de clôturer la démarche de comparaison entre méthodes qui avait été introduite précédemment. La conclusion de cette dernière est que la méthode de Newton reste la plus performante sur notre exemple, la fonction sinus, puisqu'elle converge en 4 itérations contre 6 pour la méthode de la sécante et 17 pour celle de dichotomie.

En finalité, les objectifs d'apprentissage qui sont travaillés dans ces exercices sont

- Décrire des algorithmes en pseudo-code ;
- Réaliser des algorithmes avec un logiciel adapté ;
- S'interroger sur l'efficacité d'un algorithme ;
- Écrire en instructions élémentaires une formule permettant un calcul ;

- Écrire un programme calculant et donnant la valeur d'une fonction, ainsi que les instructions d'entrée et de sortie nécessaires au traitement ;
- Programmer un calcul itératif, le nombre de répétition étant donné ;
- Programmer une instruction conditionnelle, un calcul itératif avec une fin de boucle conditionnelle.

Autrement dit, presque la totalité de ceux que nous avons annoncés dans notre section « Information sur la séquence ».

Lors de la résolution de ces exercices, les élèves risquent de se heurter à plusieurs difficultés. En effet, puisque les méthodes que nous présentons se complexifient au fur et à mesure, il en va de même pour la double transposition qui s'opère entre la présentation mathématique de ces dernières, les algorithmes informatisés associés et les programmes. Cela se traduit ici par l'augmentation du nombre de variables informatiques manipulées (x_0, x_1 et x_2) et par l'expression algébrique $x_{n+1} = x_n - f(x_n)\dfrac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$ qui est plus difficile à appréhender dans sa forme plane (voir figure 5.8).

5.3.3 Applications

Parmi les différents « exemples de recherche de zéro » proposés dans l'activité « Introduction à Python – Analyse »[IREM de Paris, 2018], nous avons retenus les deux premiers tout en intégrant des éléments du troisième. Ces derniers sont l'application de nos méthodes algorithmiques à une fonction polynomiale et à la fonction de Bolzano. Les corrections de ces applications sont trop longues pour être placées ici, elles sont cependant disponibles en annexe, dans le support de cours de la séquence version enseignant.

Ces applications sont moins liées à l'apprentissage de la programmation qu'à la justification de l'introduction de nos méthodes algorithmes. Autrement dit, il est question ici d'obtenir une réponse à notre problème qui est la recherche de zéro d'une fonction donnée. Dans cette dernière partie, il est donc question de tester ces méthodes sur différents exemples et ainsi constater leur efficacité au sens large du terme. Précisons qu'il n'est pas question ici de preuve de correction ou de terminaison de nos algorithmes. Pour fournir de telles preuves il serait par ailleurs tout à fait insuffisant de se baser uniquement sur quelques exécutions de nos algorithmes pour des instances différentes. Cela signifie que, tout au long de notre séquence, l'algorithme est uniquement considéré comme un outil au service des mathématiques, un moyen de résoudre un problème.

```
1 def secante_iteration(f,x0,x1):
2     x2 = x1 - (x1-x0)/(f(x1)-f(x0))*f(x1)
3     return x2
```

Figure 5.8 – Code Python de la fonction `secante_iteration`.

Notons par ailleurs que le fait de considérer d'autres fonctions que la fonction sinus constitue un saut informationnel. En effet, la modification de la valeur de cette variable didactique peut impliquer un changement dans les stratégies possibles de résolution de la tâche. En effet, si chacune de nos méthodes pouvait être appliquée sur la fonction sinus, nous verrons que ce n'est pas forcément le cas pour les fonctions dont il est question ici.

Nous allons voir que dans ces deux applications, les élèves sont amenés non seulement à rattacher leurs connaissances à des problèmes semblables mais aussi à distinguer dans les énoncés les informations utiles ou les attentes implicites ainsi qu'à se questionner sur les résultats obtenus. Pour ce faire, nous nous trouvons à nouveau dans une situation où les élèves travaillent de leur côté et sont aidés ponctuellement par l'enseignant jusqu'à la correction collective.

Fonction polynomiale

La fonction polynomiale P de la première application est

$$P = x^5 - 3x^3 + 1.$$

La séquence de l'IREM demande succinctement de rechercher « une valeur approchée de toutes les racines réelles de P à 10^{-9} près ». Nous allons détailler ici les stratégies de résolution possibles de cet exercice.

Tout d'abord, il est intéressant d'identifier de façon visuelle les différentes racines de la fonction P. En effet, comment approximer numériquement ces dernières à l'aide de nos algorithmes si nous n'avons pas la moindre idée de celles que nous recherchons ? Pour ce faire, les élèves doivent se servir de la fonction `graphe`. Pour cela, il faut définir au préalable une fonction correspondant au polynôme P, déterminer la fenêtre de représentation graphique en fixant les points a et b, passer ces éléments en arguments de la fonction `graphe` et l'exécuter. Il est probable que les élèves ne choisissent pas le bon intervalle de représentation à la première tentative, l'enseignant peut alors les rediriger et éventuellement indiquer le nombre de racines qu'ils sont censés déterminer. [-2 ; 2]

Ensuite, il faut appliquer au choix l'une des méthodes algorithmiques. À la vue des résultats que obtenus en termes de précision et de vitesse de convergence dans la partie précédente de la séquence, il semble plus logique de se tourner soit vers la méthode de Newton, soit vers la méthode de la sécante. Par ailleurs, puisqu'une précision de 10^{-9} est demandée dans l'énoncé, ce sont les versions conditionnelles de ces algorithmes qu'il faut utiliser. À partir de ce là, c'est aux élèves de déterminer le ou les points de départ en fonction du zéro recherché et de la méthode choisie, et ainsi de suite pour les trois racines à approcher. Il est à noter que la méthode de Newton et celle de la sécante permettent d'obtenir les mêmes approximations dans ce cas précis.

Lors de la correction de l'exercice, il est intéressant de se pencher plus particulièrement sur les solutions des élèves ayant décidé d'utiliser la méthode de Newton. En effet, en comparant avec les approximations obtenues par la méthode de la sécante, la classe devrait se rendre compte que le choix du point de départ pour la méthode de Newton influe sur la

racine vers laquelle celle-ci va converger. De fait, en fonction du point choisi, la méthode de Newton ne converge pas forcément vers la racine ciblée au départ. Après réflexion avec la classe, le professeur doit conclure que la convergence de la méthode de Newton vers le zéro souhaité nécessite que cette dernière démarre d'un point qui en est suffisamment proche. Cette constatation est abordée par l'IREM d'une manière différente, avec une autre fonction mais cette première application s'y prête tout autant.

Fonction de Bolzano

La deuxième application concerne une fonction « monstre » de l'analyse, la fonction de Bolzano. Les deux questions sur cette fonction ont attrait à son caractère continu mais non dérivable, ce qui implique que toutes les méthodes algorithmiques qui ont été vues ne peuvent s'appliquer. En effet, puisque nous n'avons pas accès à la dérivée de la fonction, la méthode de Newton ne peut être utilisée bien qu'elle s'est avérée être la plus efficace en termes de vitesse de convergence et d'approximation auparavant.

Le code en Python associé à cette fonction étant particulièrement long et complexe, ici comme dans la séquence de l'IREM, le code Python est donné ainsi que son graphe sur l'intervalle $[0, 1]$. Le code de la fonction peut être recopié de façon consciencieuse ou téléchargé à partir d'un fichier extérieur.

Il est demandé aux élèves d'approcher une des racines de cette fonction autour du point 0.2, sachant que par sa nature la fonction de Bolzano possède de nombreuses racines. En toute logique, les élèves vont pouvoir s'en rendre compte lors de leurs essais ainsi que lors de la correction par le professeur. Les résultats des élèves dépendront des points de départ qu'ils auront considérés.

5.3.4 Conclusion de l'analyse

En conclusion de cette analyse, nous pouvons dégager les quelques points qui nous semblent les plus importants, les éléments les plus marquants.

Le premier point est dédié aux algorithmes et à la façon dont ils sont présentés. Comme nous avons pu le voir, cette séquence considère les algorithmes comme des outils au service des mathématiques, des moyens de résolution de problème systématiques soumettant une réponse à ce problème en un temps fini. Rappelons-nous, les différents algorithmes associés aux méthodes de dichotomie, de Newton et de la sécante ne sont jamais présentés que comme des méthodes algorithmiques permettant de déterminer une valeur approchée d'un zéro d'une fonction. Cependant, ce constat n'est pas inattendu dans la mesure où notre séquence source est française, et que celle que nous venons de présenter en a conservé en majorité le contenu et la structure. En effet, nous avons pu nous rendre compte dans le chapitre 2 qu'il s'agit là d'une vision de l'algorithme majoritairement véhiculée par les programmes en France. Nous pouvons aussi signifier le fait qu'il n'est jamais explicitement question dans cette séquence d'algorithme-objet.

Le second point est que cette séquence recouvre à elle seule l'ensemble des structures algorithmiques que nous avions regroupées en quatre classes dans le chapitre 1. En

la parcourant nous avons rencontré des situations mettant en scène des algorithmes en pseudo-code mais aussi sous la forme de programmes, et ces derniers ont permis d'aborder à la fois l'affectation de variable, les instructions d'écriture, les structures conditionnelles de test et les structures itératives, à la fois des boucles de type WHILE que des boucles de type FOR. Il s'agit donc d'une séquence très complète de ce point de vue.

Le troisième point concerne les situations rencontrées dans la séquence. Elles sont de plusieurs types et s'agencent de telle sorte que le contrat didactique évolue tout au long de la séquence. En effet, nous retrouvons des situations a-didactiques, avec des successions de phase d'action, de formulation et de formulation visant des connaissances précises, mais aussi des situations dans lesquelles les élèves sont amenés à réinvestir les différentes connaissances qu'ils possèdent, ou qu'ils ont acquises durant le cours, ainsi que des exercices d'écriture d'algorithmes et de programmes et des applications des différentes méthodes. Les positions de l'élève et du professeur par rapport au savoir sont changeants et les situations mises en œuvre sont variées.

Enfin, nous avons aussi pu constater que la séquence prend bel et bien en compte les différents phénomènes liés à l'environnement informatisé dans lequel elle se déroule. Que ce soit dans le support de cours lui-même ou lors des phases d'institutionnalisation, des remarques et des point d'attention concernent, de manière explicite ou non, les difficultés que les élèves sont susceptibles de rencontrer lors de leur utilisation du dispositif informatique. Ces dernières pouvant être de l'ordre de l'instrumentation, de la transposition informatique (simple ou double) ou de la distance entre l'environnement papier-crayon et l'environnement informatisé.

5.4 Suites possibles de la séquence

La séquence de cours telle qu'elle est construite ici amène naturellement à se poser des questions importantes du point de vue de l'algorithmique mais aussi du point de vue des mathématiques, nous avons pu nous en rendre compte durant son analyse. La première partie de la séquence, par exemple, introduit l'idée de résolution exacte et approchée, sans plus de développement. Or, ces notions mènent logiquement à celle d'erreur d'approximation, que l'on retrouve de manière implicite dans la séquence, et par extension aux notions mathématiques d'erreur relative et absolue. Dans les faits, il est tout à fait possible d'adapter la séquence de façon à intégrer ces conceptions. Ces dernières permettraient de justifier de façon plus formelle le passage d'une méthode algorithmique à une autre puisque, nous l'avons vu, la progression de la séquence est telle que les méthodes sont abordées non seulement de la plus simple à la plus complexe mais aussi de l'erreur d'approximation la plus élevée à la plus faible. Enfin, il est aussi imaginable d'incorporer une réflexion autour de la complexité des algorithmes ou encore des preuves de terminaison et de correction. Cela reviendrait à considérer ces algorithmes non plus seulement en tant qu'outils au service des mathématiques, mais aussi en tant qu'objets à part entière.

Conclusion

Dans ce mémoire nous nous sommes intéressés de près à l'algorithmique et la programmation ainsi qu'à ses interactions avec l'enseignement des mathématiques. Tout d'abord, nous avons discuté de ce que désigne le mot « algorithmique » ainsi que des liens qu'entretient cette discipline avec la programmation et, bien évidemment, avec les mathématiques. Cette première approche du concept d'algorithme nous a permis de mettre en évidence les caractéristiques principales d'un algorithme ainsi que l'intérêt d'étudier ces objets sans se limiter aux considérations liées à leur statut de méthode de résolution de problèmes.

Ensuite, nous avons discuté de l'enseignement de l'algorithmique en France. Nous avons identifié les raisons qui ont poussé l'hexagone à intégrer de l'algorithmique et de la programmation dans ses programmes et nous avons pu constater que ces modifications sont en partie basée sur l'évolution même de notre société, dans laquelle nous nous retrouvons de plus en plus souvent confrontés à des objets et des outils informatiques. En complément de ces questionnements, nous avons analysé les programmes français, ce qui nous a permis de constater que l'algorithmique est présente de façon transversale à tous les contenus en mathématiques mais est aussi étudiée dans d'autres types de cours. De là nous avons dégagé les différentes visions de l'algorithme et avons conclu que ce dernier était majoritairement considéré comme un outil au service des mathématiques, bien que parfois considéré en tant qu'objet intéressant à étudier en tant que tel.

En prévision de l'analyse d'une séquence de cours portant sur l'algorithmique et la programmation, nous avons défini deux cadres théoriques complémentaires à celui de l'algorithmique, l'un sur la didactique des mathématiques et l'autre centré sur la didactique des TICE, autrement dit sur les nombreux effets de l'utilisation de ces nouvelles technologies pour l'enseignement sur l'apprentissage, sur les utilisateurs ainsi que sur les savoirs en eux-mêmes.

Pour finir, nous nous sommes concentrés sur l'étude à priori d'une séquence de cours portant sur l'algorithmique et la programmation en Python, basée sur une proposition de l'IREM de Paris ayant trait au domaine de l'analyse mathématique. En amont de cette analyse, nous avons supposé que nous nous trouvions dans un cas « idéal » en ce qui concerne l'algorithmique dans l'enseignement, c'est à dire que notre séquence était destinée à un public possédant déjà des notions d'algorithmique. En nous reposant sur les apports théoriques des chapitres 1, 2 et 3 nous avons étudié le déroulé de cette séquence et identifié les savoirs algorithmiques en jeu ainsi que leur traitement. Nous en avons notamment conclu que cette séquence permet d'aborder une grande majorité des apprentissages visés par la France en matière d'algorithmique et de programmation au

lycée et de travailler un certain nombre de compétences qui y sont associées. De plus, cette dernière prend bel et bien en considération les différents effets de l'utilisation d'un ordinateur et d'un langage de programmation expert tel que Python sur les savoirs, sur les élèves et sur leur apprentissage. Autrement dit, nous avons justifié le bien fondé de notre séquence tant au niveau des objectifs d'apprentissage que dans la manière d'atteindre ces objectifs.

Cependant, il s'agit là uniquement d'une analyse à priori, cela signifie que nous ne considérons pas la mise en œuvre de la séquence. Autrement dit, étant donné que la séquence n'a pas été testée en conditions réelles, nous ne pouvons que supposer de sa bonne réception par les élèves. Par conséquent, une perspective possible de cette étude serait de tester la séquence auprès d'un vrai public, et cela en vue d'un contrôle à posteriori de son fonctionnement basé sur les faits observés. Une telle analyse réflexive permettrait de mettre en lumière les faiblesses, les dysfonctionnements ou les insuffisances potentielles du cours, mais aussi éventuellement ses points forts.

Bibliographie

[Artigue, 1997] Artigue, M. (1997). *Le logiciel "Derive" comme révélateur de phénomènes didactiques liés à l'utilisation d'environnements informatiques pour l'apprentissage.* Educational studies in Mathematics, 33(2), pp.133–169.

[Balacheff, 1993] Balacheff, N. (1993). *La transposition informatique, un nouveau problème pour la didactique.* Colloque « Vingt ans de didactique des mathématiques en France », Paris, France. pp.364-370. hal-00190646.

[Blanchard and Dumont, 2019] Blanchard, J. and Dumont, M. (Année académique 2018-2019). *Notes sur le langage de programmation Python - Travaux Pratiques de Programmation [SMATM110].* Université de Namur.

[Briand, 1991] Briand, J. (1991). *Documents pour la formation des professeurs d'école en didactique des mathématiques*, chapter « Glossaire de didactique ». Actes du stage de Cahors.

[Briant, 2013] Briant, N. (2013). *Étude didactique de la reprise de l'algèbre par l'introduction de l'algorithmique au niveau de la classe de seconde du lycée français. Histoires et perspectives sur les mathématiques.* Thèse de doctorat, Université de Montperllier II - Sciences et Techniques du Languedoc. Français. tel-00920506.

[Brousseau, 1997] Brousseau, G. (1997). *La théorie des situations didactiques.* Cours donné lors de l'attribution à Guy Brousseau du titre de Docteur Honoris Causa à l'Université de Montréal.

[Brousseau, 1998] Brousseau, G. (1998). *Théorie des situations didactiques : Didactique des mathématiques 1970-1990.* La pensée sauvage Grenoble.

[Brousseau, 2010] Brousseau, G. (2010). *Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques.* Disponible sur `http://guybrousseau.com/wpcontent/uploads/2010/09/Glossaire_V5.pdf`.

[Chevallard, 1982] Chevallard, Y. (1982). *Pourquoi la transposition didactique ?* Paru dans les Actes de l'année 1981-1982, pp. 167-194. Communication au Séminaire de didactique et de pédagogie des mathématiques de l'IMAG, Université de Grenoble.

[Chevallard et al., 1985] Chevallard, Y., Johsua, M.-A., et al. (1985). *La transposition didactique : du savoir savant au savoir enseigné.* La Pensée sauvage.

[Claver, 2013] Claver, N. (2013). *Approche instrumentale et didactiques : apports de Pierre Rabardel.* Disponible sur `http://www.adjectif.net/spip/spip.php?article202`.

[Clerc et al., 2006] Clerc, J.-B., Minder, P., and Roduit, G. (2006). *La transposition didactique.* Disponible sur `http://lyonelkaufmann.ch/histoire/MHS31Docs/Seance1/TranspositionDidactique.pdf`.

[Douady, 1984] Douady, R. (1984). *Jeux de cadres et dialectiques outil-objet dans l'enseignement des Mathématiques. Une réalisation dans tout le cursus primaire.* PhD thesis.

[Henry, 2020] Henry, V. (Année académique 2019-2020). *Concepts fondamentaux de la didactique des mathématiques.* Cours de Didactique et épistémologie des mathématiques. Université de Namur.

[IREM de Paris, 2018] IREM de Paris (2018). *Introduction à Python - Analyse : Approximation d'un zéro d'une fonction réelle.* Disponible sur `https://irem.u-paris.fr/stage-algorithmique-et-programmation-au-lycee-2018-2019`.

[Johsua and Dupin, 1993] Johsua, S. and Dupin, J.-J. (1993). *Introduction à la didactique des sciences et des mathématiques*, volume 327. Presses universitaires de France Paris.

[Kahane, 2000] Kahane, J.-P. (2000). *Informatique et enseignement des mathématiques.* Rapport officiel, Commission de réflexion sur l'enseignement des mathématiques.

[Kuzniak, 2005] Kuzniak, A. (2005). *La théorie des situations didactiques de Brousseau.* Repères Irem, 61.

[MENJ, 2009a] MENJ (2009a). *Programme de mathématiques, enseignement commun, seconde générale et technologique.* Bulletin officiel, Direction générale de l'enseignement scolaire. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[MENJ, 2009b] MENJ (2009b). *Ressources pour la classe de seconde — Algorithmique —.* Document ressource. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[MENJ, 2010a] MENJ (2010a). *Programme d'enseignement spécifique de mathématiques en classe de première de la série économique et sociale et d'enseignement obligatoire au choix en classe de première de la série littéraire.* Bulletin officiel, Direction générale de l'enseignement scolaire. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[MENJ, 2010b] MENJ (2010b). *Programme d'enseignement spécifique de mathématiques en classe de première de la série scientifique.* Bulletin officiel, Direction générale de l'enseignement scolaire. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[MENJ, 2011a] MENJ (2011a). *Programme de l'enseignement spécifique et de spécialité de mathématiques de la série économique et sociale et de l'enseignement de spécialité de mathématiques de la série littéraire.* Bulletin officiel, Direction générale de l'enseignement scolaire. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[MENJ, 2011b] MENJ (2011b). *Programme d'enseignement spécifique et de spécialité de mathématiques en classe de terminale de la série scientifique.* Bulletin officiel, Direction générale de l'enseignement scolaire. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[MENJ, 2017] MENJ (2017). *Programme d'enseignement de spécialité d'informatique et sciences du numérique en classe terminale de la série scientifique.* Bulletin officiel, Direction générale de l'enseignement scolaire. (© Ministère de l'éducation nationale et de la jeunesse > `www.education.gouv.fr`).

[MENJ, 2018] MENJ (2018). *Programme d'enseignement du cycle des approfondissements (cycle 4).* Bulletin officiel, Direction générale de l'enseignement scolaire. (Ministère de l'éducation nationale et de la jeunesse > © `www.education.gouv.fr`).

[Meurist, 2014] Meurist, A. (2014). *Introduire des éléments d'algorithmique dans un cours de mathématiques : une expérience dans l'enseignement secondaire belge.* Mémoire de master, Université de Mons.

[Modeste, 2012] Modeste, S. (2012). *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve ?* Thèse de doctorat, Université de Grenoble.

[Perrenoud, 1998] Perrenoud, P. (1998). *La transposition didactique à partir de pratiques : des savoirs aux compétences.* Revue des sciences de l'éducation, 24(3), pp.487–514.

[Rabardel, 1995] Rabardel, P. (1995). *Les hommes et les technologies ; approche cognitive des instruments contemporains.*

[Robert et al., 1999] Robert, A., Lattuati, M., and Penninckx, J. (1999). *L'enseignement des mathématiques au lycée : un point de vue didactique.* Ellipses.

[Trouche, 2004] Trouche, L. (2004). *Environnements Informatisés et Mathématiques : quels usages pour quels apprentissages ?* Educational Studies in Mathematics, 55(1-3), pp.181–197.

[Université de Paris, sd] Université de Paris (s.d.). Groupe IREM Algorithmique. `https://irem.u-paris.fr/algorithmique`.

[Vanhoof, 2015] Vanhoof, W. (Année académique 2014-2015). *Cours de Programmation I [SINFB103].* Université de Namur.

[Wallonie-Bruxelles Enseignement, 2014] Wallonie-Bruxelles Enseignement (16 janvier 2014). *Compétences terminales et savoirs requis - humanités générales et technologiques – mathématiques.* Bulletin officiel, Fédération Wallonie-Bruxelles.

Annexe A

Support de cours – Version enseignant

Approximation d'un zéro d'une fonction réelle : approche algorithmique

1 Mise en place

Afin de vous aider tout au long de ce cours, vous avez à disposition en fin de chapitre un aide-mémoire du Python.

1.1 Importer les modules

Dans ce cours nous allons manipuler des fonctions mathématiques au travers du langage de programmation **Python**. Pour ce faire, nous avons besoin

- du module intitulé `math` qui donne accès à des fonctions mathématiques prédéfinies telles que le sinus, le cosinus, la racine carrée, etc.
- du module `matplotlib` qui permet de représenter graphiquement des fonctions.

1.2 Définition de la fonction sinus

Nous allons travailler sur la fonction mathématique $x \to \sin(x)$.
Afin de pouvoir l'utiliser dans la suite, nous la définissons en Python comme suit.

```
1 def fsin(x) :
2         y = sin(x)
3         return y
```

1.3 Représentation graphique

Grâce au module `matplotlib` que nous avons importé, nous pouvons définir une fonction Python permettant de tracer le graphe d'une fonction f sur un certain intervalle $[a, b]$.

```
1  def graphe(f,a,b) :
2          # axe des abscisses en 0
3          axes = plt.gca()
4          axes.spines['bottom'].set_position(('data',0))
5          # liste des abscisses
6          x = [a + i/100*(b-a) for i in range(100)]
7          # liste des ordonnées
8          y = [f(u) for u n x]
9          # affichage du graphe
10         plt.show(x,y)
11         plt.show()
```

Remarque (Notation)

L'écriture suivante correspond à une *mutation de liste*.

```
[a + i/100*(b-a) for i in range(100)]
```

Il s'agit d'une notation plus simple et concise permettant de construire la liste des nombres de la forme $a+\frac{1}{100}(b-a)$ pour i allant de 0 à 99. En mathématique, cette écriture correspond à la suite notée

$$\left(a+\frac{1}{100}(b-a)\right)_{0\leq i<100},$$

ou encore à la notation ensembliste

$$\left\{a+\frac{1}{100}(b-a) \mid 0\leq i<100\right\}.$$

1. Représentez le graphe de la fonction sinus sur l'intervalle $[-4,4]$.

```
graphe(fsin,-4,4)
```

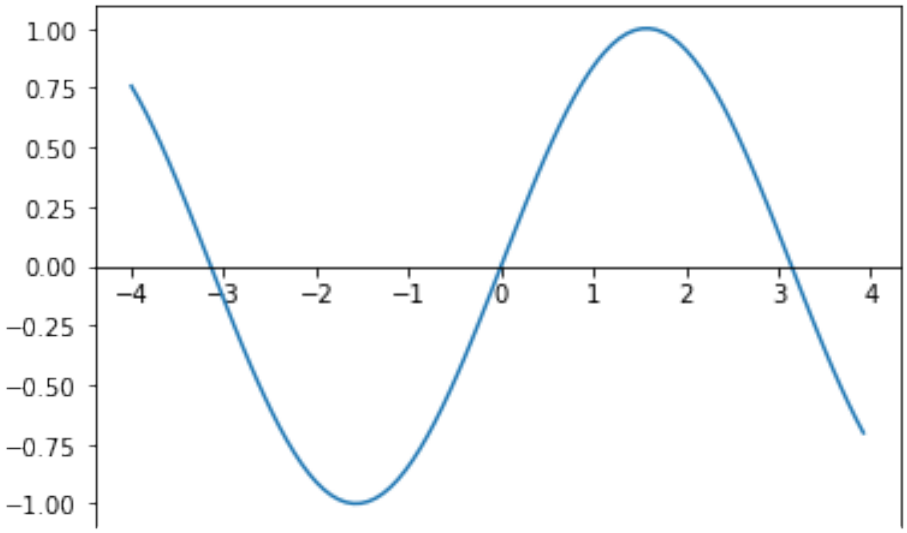
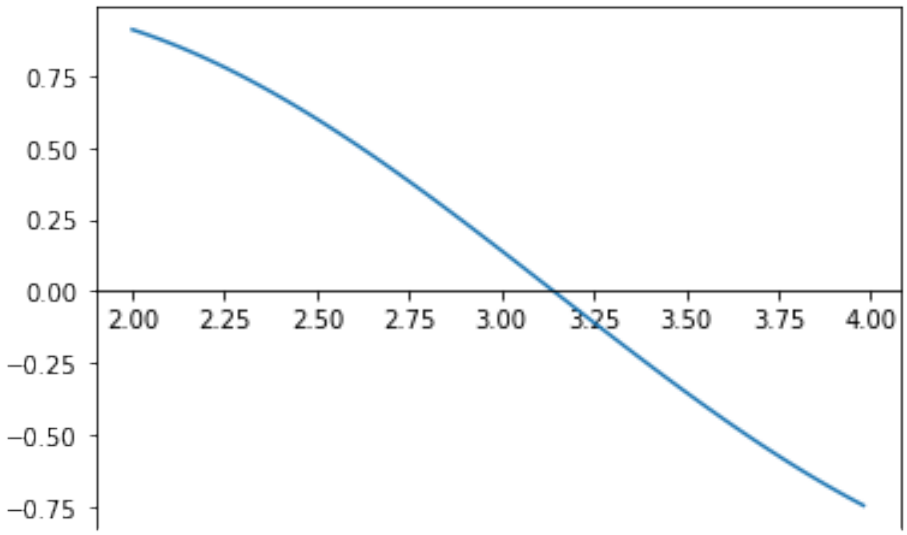


FIGURE 1 – Graphe de la fonction $x \to sin(x)$ sur l'intervalle $[-4;4]$.

2. Repérez à partir du graphe ci-dessus les racines de la fonction sinus sur l'intervalle $[-\pi,\pi]$. racine en $x=-\pi$, $x=0$ et $x=\pi$

3. Déterminez analytiquement les racines de la fonction sinus sur l'intervalle $[-\pi,\pi]$. $\sin(x)=0 \Leftrightarrow x=\pi+k\pi$, $k=\{-2,-1,0\}$

4. Nous nous intéressons à la racine comprise entre les abscisses 2 et 4. Représentez le graphe de la fonction sinus sur cet intervalle.



1.4 Résolution exacte et approchée

Comme nous venons de le rappeler, déterminer la/les racine(s) d'une fonction f revient à

résoudre l'équation $f(x) = 0$.

Pour une fonction telle que celle du sinus, la résolution **exacte** de cette équation peut se faire rapidement et sans trop de difficultés... *Mais ce n'est pas toujours le cas.*

Pourquoi ?

- ▷ Dans la pratique, la tâche peut s'avérer bien plus complexe et il arrive que l'équation ne puisse être résolue exactement.

  ⇢ Si la fonction est une composée de fonctions usuelles complexe par exemple.

- ▷ Certains théorèmes permettent d'assurer l'existence de solutions à ce type d'équation (voire leur unicité) mais ne donnent pas d'information sur leur valeur.

Comment y remédier ?

En utilisant une **méthode algorithmique** afin de résoudre notre problème général :

déterminer une **valeur approchée** de la/les racine(s) recherchée(s) d'une fonction donnée.

2 Approche algorithmique

Dans cette deuxième section, nous allons explorer différentes méthodes algorithmiques permettant de répondre à la question que nous venons de poser. Autrement dit, des algorithmes qui permettent de d'obtenir une solution approchée d'un zéro d'une fonction réelle. Nous étudierons trois méthodes et nous les implémenterons en langage Python afin de les tester sur la fonction sinus, entre autres.

2.1 Méthode de dichotomie

La première méthode que nous allons aborder est fondée sur le théorème des valeurs intermédiaires (que vous connaissez déjà).

Théorème (Théorème des valeurs intermédiaires)

Soit f une fonction continue sur un intervalle $[a, b]$. Alors,

$\forall\, k \in \mathbb{R}$ strictement compris entre $f(a)$ et $f(b)$, $\exists\, c \in]a, b[$ tel que $f(c) = k$.

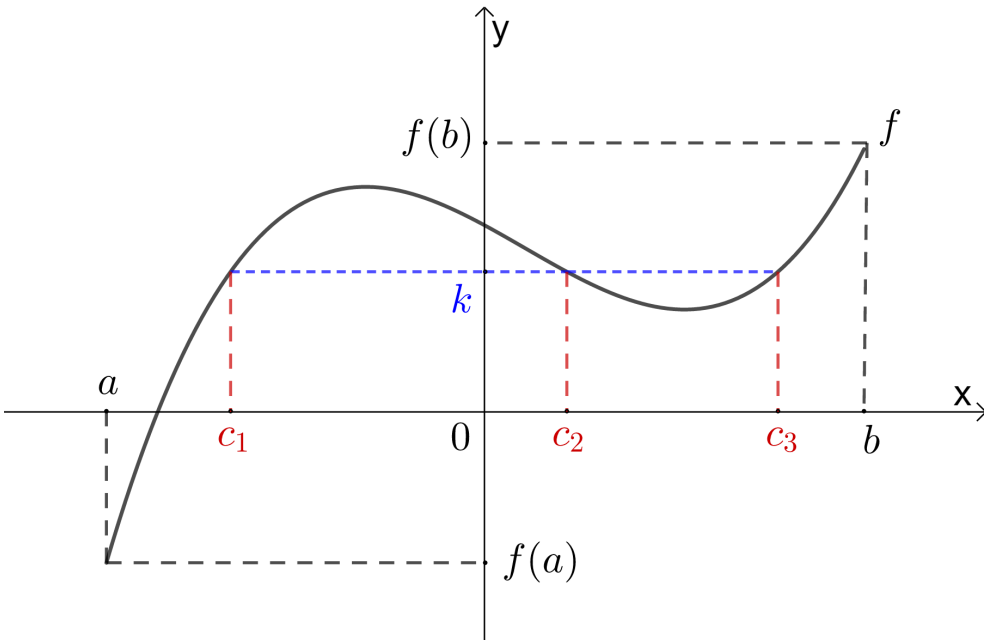


FIGURE 2 – Illustration du théorème des valeurs intermédiaires.

Dans notre cas Nous sommes à la recherche d'une racine d'une fonction dans un intervalle donné, autrement dit nous recherchons

$$\text{le réel } c \text{ appartenant à }]a, b[\text{ tel que } f(c) = 0.$$

Grâce au théorème des valeurs intermédiaires, nous pouvons déduire qu'une telle situation se présente lorsque $f(a)$ et $f(b)$ sont non nuls et de signes opposés.

L'idée de base de l'approche par dichotomie est simple. En effet, elle consiste à couper l'intervalle $]a, b[$ en deux en considérant le point milieu $m = \frac{(a+b)}{2}$.

Dès lors, plusieurs cas sont possibles

1. si $f(m) = 0$, alors nous avons trouvé une racine de f dans l'intervalle $]a, b[$,
2. si $f(a)$ et $f(m)$ sont non nuls et de signes opposés, alors la recherche continue dans l'intervalle $]a, m[$,
3. sinon, $f(m)$ et $f(b)$ non nuls et de signes opposés, alors la recherche continue dans l'intervalle $]m, b[$.

2.1.1 Algorithme

Écrivez l'algorithme associé à cette méthode.

Algorithme 2.1 : Une itération de la méthode de dichotomie

Entrées : f une fonction continue, a, b deux nombres encadrant une racine de f
m est le milieu du segment $]a, b[$
si $f(m) = 0$ **alors**
    Sorties : m
sinon si $f(a) * f(m) < 0$ **alors**
Sorties : l'intervalle $]a, m[$
sinon
    Sorties : l'intervalle $]m, b[$
fin

Fonction en Python

Analysez la fonction en Python ci-dessous qui prend en entrée une fonction f ainsi que les nombres a et b qui encadrent une racine de cette fonction.

```
1  def dichotomie_iter(f,a,b):
2      m = (a+b)/2
3      if f(m) == 0 :
4          a = m
5          b = m
6          return a, b
7      if f(a)*f(m) < 0 :
8          b = m
9          return a, b
10     else :
11         a = m
12         return a, b
```

1. Que renvoie cette fonction ?

   L'intervalle sur lequel la recherche dichotomique doit continuer ou la racine qui a été trouvée.

2. Qu'en déduisez-vous ?

   Il s'agit du code de la fonction Python correspondant à l'algorithme qui a été complété juste avant.

Exercices

Testez la fonction `dichotomie_iter` sur la fonction sinus que nous avons définie, sur $]3, 4[$.

```
1 dichotomie_iter(fsin,3,4)
```

1. Quel intervalle obtenez-vous ? Déterminez-en le milieu. Intuitivement, est-ce une bonne approximation du zéro recherché ? Pourquoi ?

   $]3, 3.5[$ dont le milieu est $m = 3.25$ À eux d'interpréter, ils doivent fournir des explications en français.

2. Quelle stratégie permettrait une meilleure approximation de cette racine ?

   La solution la plus simple qui vienne en tête est d'effectuer plusieurs itérations, la seconde est d'imposer la longueur de l'intervalle d'arrivée, donc une certaine précision (mais cette dernière ne viendrait probablement pas à l'esprit des élèves).

2.1.2 Version itérative

La fonction `dichotomie_iter` que nous venons d'analyser correspond à une seule itération de la méthode de dichotomie, ce qui ne nous permet pas d'obtenir une approximation satisfaisante de la racine que nous recherchons.

Pour obtenir un meilleur encadrement de la racine (et donc à une meilleure approximation de cette dernière), nous pouvons par exemple itérer la fonction précédente un certain nombre de fois, effectuer une boucle.

Algorithme

Écrivez l'algorithme qui prend en entrées la fonction f, les nombres a et b encadrant un zéro de f, le nombre n de pas/d'itérations souhaité(e)s, et qui renvoie l'approximation du zéro de f dans l'intervalle $]a, b[$ obtenue après les n itérations.

Algorithme 2.2 : Méthode de dichotomie - version itérative

1 **Entrées :** f une fonction continue, a et b deux nombres, n un entier.
2 **pour** $i = 0$ **à** $n - 1$ **faire**
3 | Appliquer la fonction `dichotomie_iter` sur l'intervalle $]a, b[$.
4 **fin**
5 Calculer le milieu m du dernier intervalle ainsi déterminé.
6 **Sorties :** m

Fonction en Python

Écrivez à présent la fonction `dichotomie_nbpas` associée à cet algorithme en Python.

```
def dichotomie_nbpas(f,a,b,n):
    for i in range(n) :
        a, b = dichotomie_iter(f,a,b)
    m = (a+b)/2
    return m
```

Exercices

À présent, testez la fonction `dichotomie_nbpas` sur la fonction sinus que nous avons définie, pour l'intervalle $]3, 4[$ et pour un nombre de pas égal à 10.

1. Quel résultat obtenez-vous ?

```
dichotomie_nbpas(fsin,3,4,10)
        # 3.14111328125
```

2.1.3 Version conditionnelle

Algorithme

Écrivez l'algorithme prenant en entrées une fonction f, deux abscisses a et b encadrant un zéro de f, $e > 0$ la précision souhaitée et renvoyant une valeur approchée d'une racine de f dans l'intervalle $]a, b[$, à e près.

Algorithme 2.3 : Méthode de dichotomie - version conditionnelle

Entrées : f une fonction continue, a et b deux nombres, $e > 0$
tant que *distance*$(a, b) > e$ **faire**
    Appliquer la fonction `dichotomie_iter` sur a et b
fin
Calculer le milieu m du premier intervalle $]a, b[$ tel que distance$(a, b) = e$
Sorties : m

Fonction en Python

Écrivez en Python la fonction `dichotomie_eps(f,a,b,e)` associée à cet algorithme.

```
def dichotomie_eps(f,a,b,e):
    # n = 0
    while abs(b - a)>e :
        a, b = dichotomie_iter(f,a,b)
        # n += 1 ou n = n+1
    m = (a+b)/2
    # print(n)
    return m
```

Exercices

1. Testez la fonction `dichotomie_eps(f,a,b,e)` sur la fonction sinus, sur l'intervalle $[3, 4]$, pour déterminer une valeur approchée de la racine à 10^{-5} près.

```
dichotomie_eps(fsin,3,4,10**(-5))
        # 3.141590118408203
```

   (a) Quel résultat obtenez-vous ? 3.141590118408203

   (b) Combien d'itérations sont nécessaires pour atteindre cette précision ? 17 itérations

2. Exécutez la fonction `dichotomie_eps(f,a,b,e)` sur `fsin`, entre 3 et 4, mais pour les précisions allant de 10^{-3} à 10^{-10}.

   (a) Combien d'itérations sont nécessaires pour atteindre la convergence dans chacun de ces cas ? Complétez le tableau suivant.

Précision	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
Nbr d'itérations	10	14	17	20	24	27	30	34

   (b) Comment évolue ce nombre d'itérations en fonction de la précision demandée ?
   Il ne fait qu'augmenter lorsque la précision demandée augmente.

Remarque (Conséquence)

Imposer une précision relativement importante a pour conséquence d'augmenter significativement le nombre d'itérations nécessaires avant d'atteindre la convergence.

2.2 Méthode de Newton

La deuxième méthode est appelée méthode de Newton. Contrairement à la méthode de dichotomie, il est nécessaire ici que nous ayons accès à la **dérivée** de la fonction f. Cette méthode est dite **itérative**. À partir d'un point x_0 suffisamment proche de la racine que l'on souhaite approximer, chaque nouveau point x_{n+1} est calculé à partir du point x_n précédent.

Le point x_{n+1} est défini comme l'abscisse du point d'intersection entre

- la **tangente** T à la courbe de f au point d'abscisse x_n et
- l'axe des abscisses.

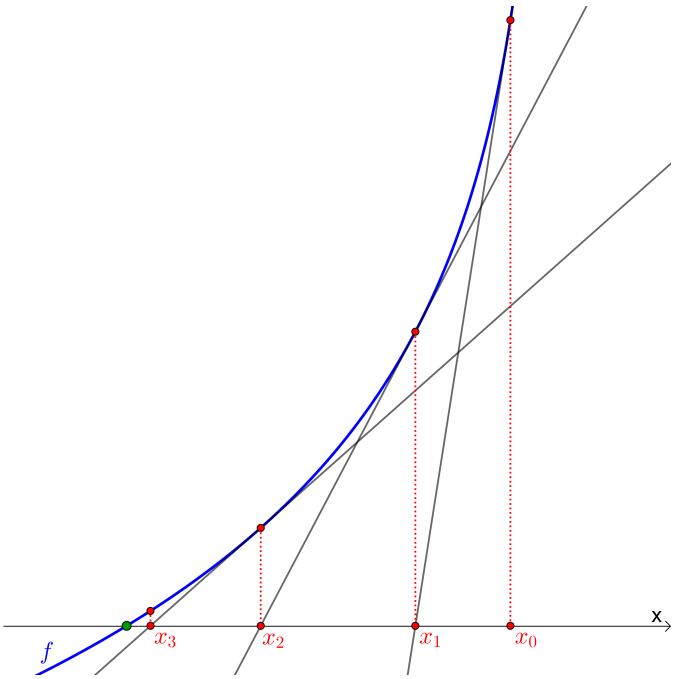


FIGURE 3 – Illustration de la méthode de Newton.

Terme général de la suite

Le point x_{n+1} peut être déterminé à partir du point précédent x_n ainsi que des valeurs de f et de f' en ce point. En effet,

- ▷ Équation de la tangente en x_n : $y = f(x_n) + f'(x_n)(x - x_n)$
- ▷ Intersection entre la tangente et l'axe des abscisses :

$$\begin{aligned} 0 &= f(x_n) + f'(x_n)(x - x_n) \\ \Leftrightarrow \quad x &= x_n - \frac{f(x_n)}{f'(x_n)} \equiv x_{n+1} \end{aligned}$$

La suite (x_n) ainsi construite converge alors vers le zéro de f proche de x_0, à condition que $f'(x_n) \neq 0$.

Fonction en Python

Comme précédemment, il est plus confortable de définir en premier lieu une fonction permettant d'effectuer une seule itération de la méthode de Newton. Ci-dessous se trouve la fonction Python `newton_iter(f,df,x0)` s'appliquant sur la fonction f et considérant la dérivée de cette fonction ainsi que le point de départ $x0$.

```
def newton_iter(f,df,x0) :
        xn = x0 - f(x0)/df(x0)
        return xn
```

2.2.1 Version itérative

Pour obtenir une meilleure approximation du zéro de notre fonction, nous avons vus les deux approches suivantes :

1. effectuer n itérations successives,
2. imposer la précision souhaitée.

Ces approches peuvent aussi s'appliquer à la méthode de Newton et la première est représentée par l'algorithme suivant.

Algorithme 2.4 : Méthode de Newton - version itérative

Entrées :
f une fonction continue et dérivable, df sa dérivée, x_0 un réel et n un entier positif.

pour $i = 0$ **à** $n - 1$ **faire**
    Appliquer la fonction `newton_iter` en x_0 pour déterminer le point x_n.
    Mise à jour de x_0
fin

Sorties : x_n

Implémentez cet algorithme en Python en définissant une fonction `newton_nbpas(f,df,x0,n)` qui effectue n itérations de la méthode de Newton sur la fonction donnée f à partir du point x_0.

```
def newton_nbpas(f,df,x0,n):
    for i in range(n) :
        xn = newton_iter(f,df,x0)
        x0 = xn
    return xn
```

2.2.2 Version conditionnelle

Écrivez l'algorithme qui retourne le premier point x_n vérifiant $|x_n - x_0| > e$ en partant de $x0$. Ensuite, implémentez cet algorithme en Python sous la forme de la fonction `newton_eps(f,df,x0,e)`.

Algorithme 2.5 : Méthode de Newton - version conditionnelle

Entrées : f une fonction continue et dérivable, df sa dérivée, x_0 un réel et $e > 0$
Appliquer la fonction `newton_iter` en x_0 pour déterminer le point x_n
tant que $|x_n - x_0| > e$ **faire**
    Mettre à jour x_0
    Appliquer la fonction `newton_iter` en x_0 pour déterminer le point x_n
fin
Sorties : x_n

```python
def newton_eps(f,df,x0,e):
    xn = newton_iter(f,df,x0)
    # n = 0
    while abs(xn-x0) > e :
        x0 = xn
        xn = newton_iter(f,df,x0)
        # n += 1 ou n = n+1
    # print(n)
    return xn
```

Exercices

1. Appliquez `newton_nbpas` à la fonction sinus à partir du point 4 et pour 10 itérations.
   (a) Qu'obtenez-vous ?

```python
newton_nbpas(fsin,fonction_cosinus,4,10)
        # 3.141592653589793
```

2. Appliquez la fonction `newton_eps` à la fonction sinus à partir du point 4 et pour une précision de 10^{-10}.

```python
newton_eps(fsin,lambda x: cos(x),4,10**(-10))
        # 3.141592653589793 pour 4 itérations
```

   (a) Combien d'itérations sont nécessaires pour obtenir la convergence ? 4
   (b) Cette convergence est-elle plus rapide ou plus lente que la méthode de dichotomie ? Beaucoup plus rapide puisque la convergence est atteinte après 17 itérations pour la méthode de dichotomie avec une précision moindre de 10^{-5}.

2.3 Méthode de la sécante

La méthode de la tangente supposais que la fonction soit dérivable, or il est aussi possible que ce ne soit pas le cas, ou que cette dernière ne soit pas accessible (comme c'est le cas si la fonction est calculée par un algorithme complexe). La méthode itérative de la sécante permet de palier à ce problème puisqu'elle ne nécessite pas l'utilisation de la dérivée de la fonction.

2.3.1 Construction

La méthode de Newton et celle de la sécante sont très similaires, à ceci près qu'elle ne considère pas la tangente à la courbe f et qu'elle nécessite de démarrer à partir de deux points au lieu d'un seul.

- ▷ Les points de départ x_0 et x_1 doivent être proches de la racine de f recherchée.
  - ⇢ ils ne doivent pas nécessairement encadrer la racine, ils peuvent se trouver tous les deux du même coté.
- ▷ À chaque itération, le point x_{n+1} est calculé à partir des points x_n et x_{n-1}
- ▷ Le point x_{n+1} est l'abscisse du point d'intersection de
  - la droite qui coupe la courbe de f aux points d'abscisses x_n et x_{n-1},
  - l'axe des abscisses.

Autrement dit, x_{n+1} est défini comme

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

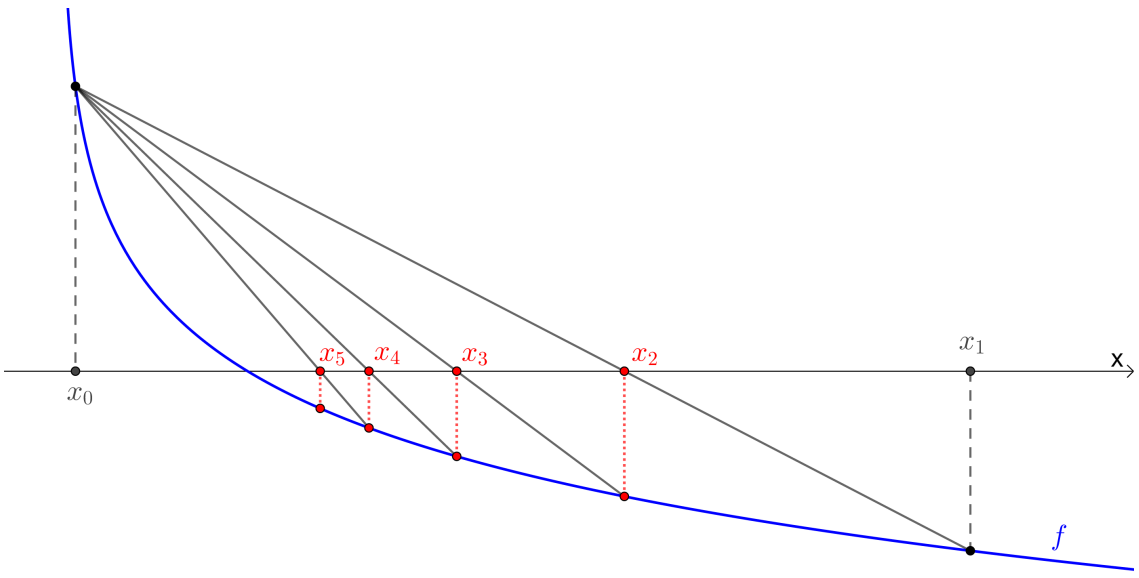


FIGURE 4 – Illustration de la méthode de la sécante.

Exercices Pourquoi n'avoir pas demandé d'écrire l'algorithme d'abord ?

1. Écrivez la fonction `secante_iteration(f,x0,x1)` qui retourne l'abscisse x_2 du point obtenu après une itération de la méthode de la sécante à partir des abscisses x_0 et x_1.

```
def secante_iteration(f,x0,x1):
    x2 = x1 - (x1-x0)/(f(x1)-f(x0))*f(x1)
    return x2
```

2. Écrivez les fonctions `secante_nbpas(f,x0,x1,n)` et `secante_eps(f,x0,x1,e)` similaires à celles décrites dans la méthode de Newton.

```
def secante_nbpas(f,x0,x1,n):
    for i in range (n):
        x2 = secante_iteration(f,x0,x1)
        x0 = x1
        x1 = x2
    return x2
```

```
def secante_eps(f,x0,x1,e):
    # n = 0
    while abs(x1-x0) > e :
        x2 = secante_iteration(f,x0,x1)
        x0 = x1
        x1 = x2
        # n += 1 ou n = n+1
    # print(n)
    return x2
```

3. Testez ces fonctions sur la fonction sinus au départ des points $x_0 = 3.9$ et $x_1 = 4$.
   (a) Quelles approximations de la racine obtenez-vous dans les deux cas ? Considérez 10 itérations pour la version itérative et une précision de 10^{-10} pour la version conditionnelle.
   3.141592653589793

```
secante_nbpas(fsin,3.9,4,10)
        # 3.141592653589793
```

```
secante_eps(fsin,3.9,4,10**(-10))
        # 3.141592653589793 pour 6 itérations
```

   (b) Après combien d'itérations la convergence est-elle atteinte pour la méthode de la sécante conditionnelle ? 6 itérations
   (c) Comparez cette vitesse de convergence avec celle des deux autres méthodes que nous avons vues, dans leur version conditionnelle. La méthode de Newton reste la plus performante sur l'exemple que nous considérons, elle converge en 4 itérations contre 6 pour la méthode de la sécante et 17 pour la dichotomie.

3 Applications

3.1 Fonction polynomiale

Considérons à présent le polynôme P suivant

$$P = x^5 - 3x^3 + 1.$$

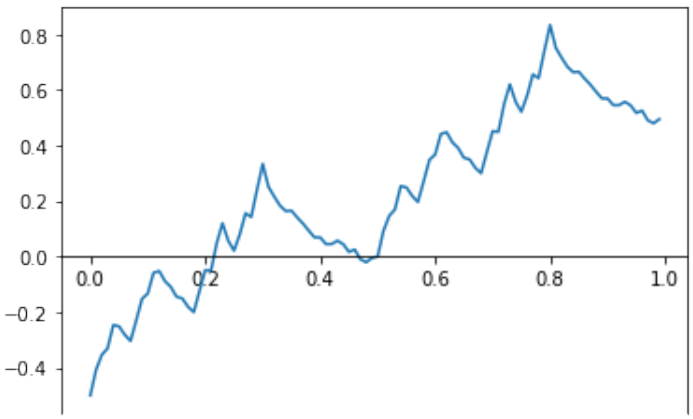
Recherchez une valeur approchée des racines réelles de ce polynôme à 10^{-9} près.

3.2 Fonction de Bolzano

La fonction de Bolzano est une fonctions « monstres » de l'analyse. Il s'agit d'une fonction fractale qui, par construction, est continue mais non dérivable. Ci-dessous se trouve le code Python permettant de définir cette fonction.

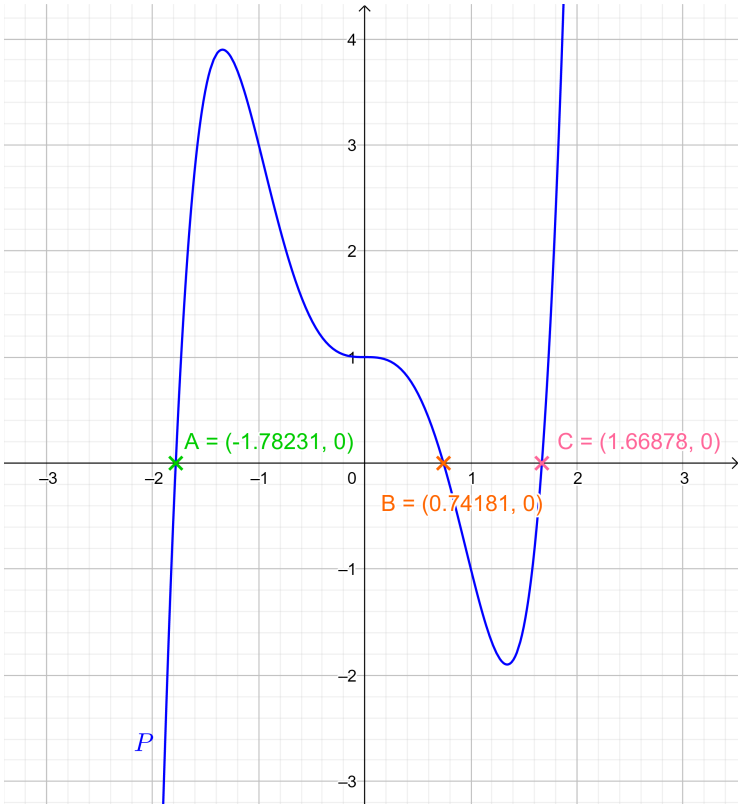
```
def bolzano(x) :
    xa,xb,ya,yb = 0,1,-0.5,0.5
    while xb-xa > 10**(-13):
        xl,yl = xb-xa,yb-ya
        if x <= xa + 3/8*xl:
            xa,xb,ya,yb=xa,xa+3/8*xl,ya,ya+5/8*yl
        elif x<=xa+1/2*xl:
            xa,xb,ya,yb=xa++3/8*xl,xa+1/2*xl,ya+5/8*yl,ya+1/2*yl
        elif x<=xa+7/8*xl:
            xa,xb,ya,yb=xa+1/2*xl,xa+7/8*xl,ya+1/2*yl,yb+1/8*yl
        else :
            xa,xb,ya,yb=xa+7/8*xl,xb,yb+1/8*yl,yb
    return (ya+yb)/2
```

```
graphe(bolzano,0,1)
```



Par construction, la fonction de Bolzano possède une racine en 0.5, mais elle possède aussi (au moins), un autre zéro autour de 0.2. Utilisez une des méthodes que nous avons vues pour approcher une racine de cette fonction autour de 0.2.

3.3 Correction fonction polynomiale :




```python
def fpol(x) :
    return (x**(5) - 3*x**(3) + 1)

graphe(fpol,-2,2)
```

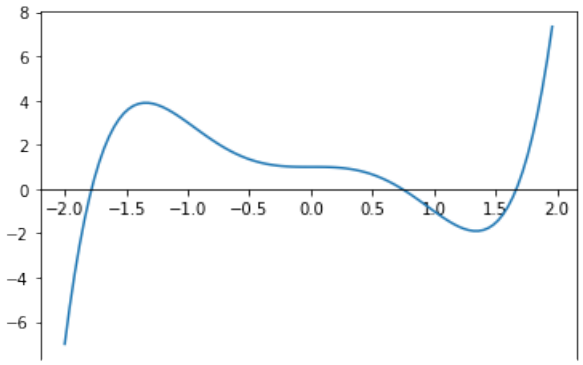
```python
# derivee
def dfpol(x):
    return (5*x**(4) - 9*x**(2))

# approximation des racines
print("Racine A")
a = dichotomie_eps(fpol,-2,0,10**(-9))
b = newton_eps(fpol,dfpol,-2,10**(-9))
c = secante_eps(fpol,-2,-1.9,10**(-9))
print("\n dichotomie :",a,"\n newton :",b,"\n secante :",c,"\n")

print("Racine B")
d = dichotomie_eps(fpol,0,1,10**(-9))
e = newton_eps(fpol,dfpol,0.5,10**(-9))
f = secante_eps(fpol,0,1,10**(-9))
print("\n dichotomie :",d,"\n newton :",e,"\n secante :",f,"\n")
```

Pourquoi les 3 méthodes en même temps ?

```
17
18 print("Racine C")
19 g = dichotomie_eps(fpol,1,2,10**(-9))
20 h = newton_eps(fpol,dfpol,1.5,10**(-9))
21 i = secante_eps(fpol,1.5,2,10**(-9))
22 print("\n dichotomie :",g,"\n newton :",h,"\n secante :",i,"\n")
```




```
Racine A

 dichotomie : (-1.7823087801225483, 31)
 newton : (-1.7823087800458841, 5)
 secante : (-1.7823087800458841, 7)

Racine B

 dichotomie : (0.7418139302171767, 30)
 newton : (0.741813930486773, 4)
 secante : (0.741813930486773, 8)

Racine C

 dichotomie : (1.668777592945844, 30)
 newton : (1.6687775925403805, 5)
 secante : (1.6687775925403805, 8)
```

En comparant avec les approximations des racines obtenues par la méthode de la sécante, nous pouvons nous rendre compte que le choix du point de départ pour la méthode de Newton influe sur la racine vers laquelle celle-ci va converger. En effet, en fonction du point choisi, la méthode de Newton ne converge pas forcément vers la racine recherchée.

Quelques exemples de ce qui peut se produire :

```
1 a = secante_eps(fpol,-2,-1.5,10**(-9))
2 b = newton_eps(fpol,dfpol,-1,10**(-9))
3 print("recherche racine A -\n secante :",a,"\n newton :",b)
```

```
recherche racine A -
 secante : (-1.782308780045885, 8)
 newton : (1.6687775925403805, 5)
```

```
1 a = secante_eps(fpol,0.5,1,10**(-9))
2 b = newton_eps(fpol,dfpol,0.3,10**(-9))
3 print("recherche racine B -\n secante :",a,"\n newton :",b)
```

```
recherche racine B -
 secante : (0.7418139304867731, 6)
 newton : (1.6687775925403807, 6)
```

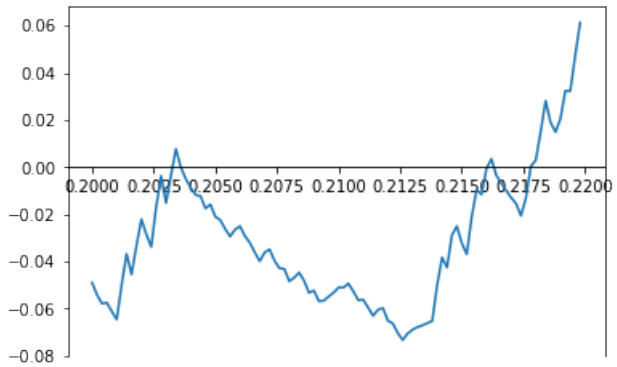
```
1 a = secante_eps(fpol,1.5,2,10**(-9))
2 b = newton_eps(fpol,dfpol,1,10**(-9))
3 print("recherche racine C -\n secante :",a,"\n newton :",b)
```

```
recherche racine C -
 secante : (1.6687775925403805, 8)
 newton : (0.741813930486773, 4)
```

Pour assurer la convergence vers le zéro souhaité, il est nécessaire que la méthode démarre d'un point assez proche de ce dernier.

3.4 Correction fonction de Bolzano :

1.



On ne peut pas utiliser la méthode de Newton car la fonction n'est pas dérivable.

```
graphe(bolzano,0.2,0.22)

# plusieurs possibilités en fonction des points de départ
# quelques-uns d'entre eux :
a = dichotomie_eps(bolzano,0.2,0.204,10**(-9))
b = dichotomie_eps(bolzano,0.2,0.205,10**(-9))
c = secante_eps(bolzano,0.217,0.2175,10**(-9))
d = secante_eps(bolzano,0.21,0.2175,10**(-9))
print(a,b,c,d)
```

$a = (0.20311868047714238, 22)$, $b = (0.20499999970197674, 23)$,
$c = (0.2177996450391367, 18)$, $d = (0.21598714548364598, 24)$

Aide-mémoire Python

1 Importer une librairie

```
from math import *
import matplotlib.pyplot as plt
```

La libraire `math` donne accès aux fonctions mathématiques prédéfinies telles que le sinus, le cosinus, la racine carrée, etc. Tandis que le module `matplotlib` permet de faire des graphiques.

2 Affectation simple

```
mavariable = -2
```

3 Opérations arithmétiques

Ci-dessous se trouvent les symboles utilisés pour les opérations arithmétiques de base.

Opération	**Symbole**	**Écriture**
Addition	+	x + y
Soustraction	-	x - y
Multiplication	*	x * y
Puissance	**	x**y
Division	/	x/y

4 Commentaire de code

```
# ceci est un commentaire
```

5 Structures de test

5.1 Test `if`

▷ **En langage naturel**

```
Si condition
        alors instruction
Fin du Si
```

▶ **En langage Python**

```
if (condition) :
        instruction
```

5.2 Test `if/then/else`

▷ **En langage naturel**

```
Si condition1
        alors instruction1
Sinon si condition2
        alors instruction2
Sinon
        instruction3
Fin du Si
```

▶ **En langage Python**

```
if (condition1) :
        instruction1

elif (condition2) :
        instruction2

else :
        instruction3
```

6 Opérations conditionnelles

Dans le tableau ci-dessous se trouvent les symboles utilisés pour les opérations conditionnelles.

Et si on met = tout seul, qu'est ce qui se passe ?

Opération	Symbole	Écriture
strictement inférieur à	<	x < y
inférieur ou égal à	<=	x <= y
strictement supérieur à	>	x > y
supérieur ou égal à	>=	x >= y
égal à	==	x == y
n'est pas égal à	!=	x != y

7 Structures d'itération

7.1 Boucle `while`

▷ **En langage naturel**

```
Tant que condition
        instructions
Fin du Tant que
```

▶ **En langage Python**

```
while (condition):
        instructions
```

7.2 Boucle `for`

▷ **En langage naturel**

```
Pour variable allant de valeur_début à valeur_fin
        instructions
Fin Pour
```

▶ **En langage Python**

```
for i in range(n) :
        instructions
```

8 Fonctions

8.1 Définition d'une fonction

```
def ma_fonction(argument1, argument2):
        instructions
        return sortie1, sortie2
```

Pourrais tu donner un exemple ?
Qu'est ce qui se cache sous "argument 1", "argument 2", "sortie 1", "sortie 2", "a" et "b" ?

8.2 Appel de fonction

```
sortie1, sortie2 = ma_fonction(a, b)
```

9 Fonctions utiles

La librairie `math` met à disposition un ensemble de fonctions Python prédéfinies bien utiles. Ci-dessous se trouve certaines d'entre elles[1].

Fonction	Utilisation	Détails
Affichage	`print(x1, x2, ...)`	Affiche la valeurs des variables `x1, x2, ...`
Valeur absolue	`abs(x)`	Détermine la valeur absolue de la variable `x`
Sinus	`sin(x)`	Retourne le sinus de `x` radians
Cosinus	`cos(x)`	Retourne le cosinus de `x` radians
Tangente	`tan(x)`	Retourne la tangente de `x` radians
Conversion en degrés	`degrees(x)`	Convertit l'angle x de radians en degrés.
Conversion en radian	`radians(x)`	Convertit l'angle x de degrés en radians.

1. L'ensemble des fonctions disponibles dans le module `math` sont disponibles dans la documentation Python : `https://docs.python.org/fr/3.5/library/math.html`

Annexe B

Séquence source

ZeroActivite

December 5, 2018

IREM de Paris – Groupe Algorithmique

1 Introduction à Python – Analyse

1.1 Approximation d'un zéro d'une fonction réelle

On va mettre en oeuvre différentes méthodes algorithmiques pour trouver une valeur approchée d'un zéro d'une fonction réelle.

On commence par importer la librairie `mathpour` avoir les fonctions mathématiques telles que la fonction `sin`, et on importe également `matplotlib` pour faire des graphiques.

```
In [1]: from math import *
        import matplotlib.pyplot as plt
```

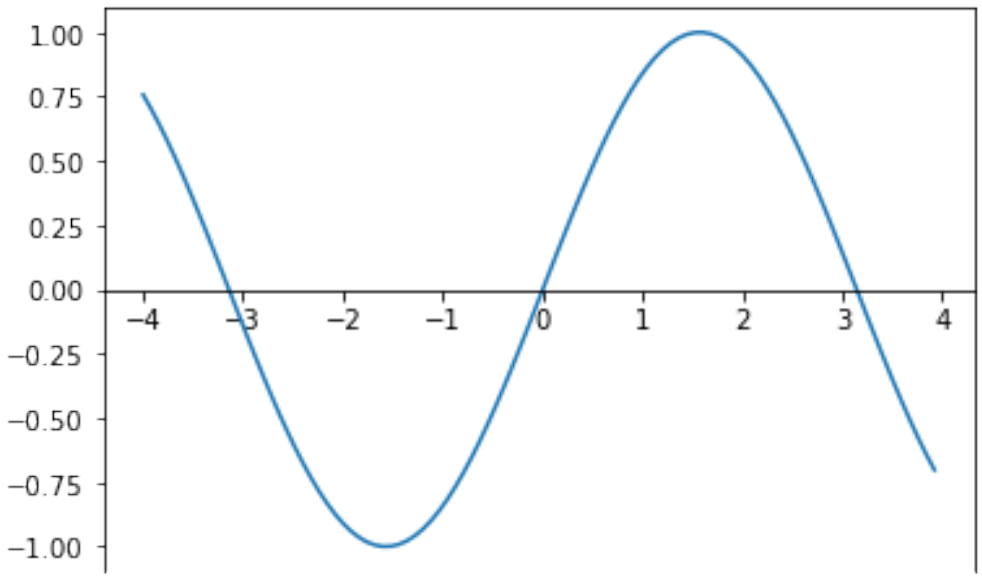
On va définir une fonction réelle f dont on recherche un ou les zéros. On utilise ici la fonction $x \mapsto \sin x$ qui est très simple et qu'on connaît bien, ce qui nous permettra de vérifier que nos algorithmes se comportent comme il faut. Nous testerons les méthodes vues ci-dessous sur d'autres exemples dans la partie D.

```
In [2]: def fsin(x):
            return sin(x)
```

On définit une fonction Python permettant de tracer le graphe d'une fonction f entre a et b.

```
In [3]: def graphe(f, a, b):
             # on place l'axe des abscisses en 0
            axes = plt.gca()
            axes.spines['bottom'].set_position(('data', 0))
            # on prépare la liste des abscisses...
            x = [a + i/100*(b-a) for i in range(100)]
            # ... et celle des ordonnées
            y = [f(u) for u in x]
            # on affiche le graphe
            plt.plot(x, y)
            plt.show()

        graphe(fsin,-4,4)
```



Remarque : L'écriture

```
[a+i/100*(b-a) for i in range(100)]
```

s'appelle une *mutation de liste*, elle sert à construire de manière concise la liste des nombres de la forme $a + \frac{i}{100}(b-a)$ pour i allant de 0 à 99. Il faut rapprocher cette notation de la notation de suite

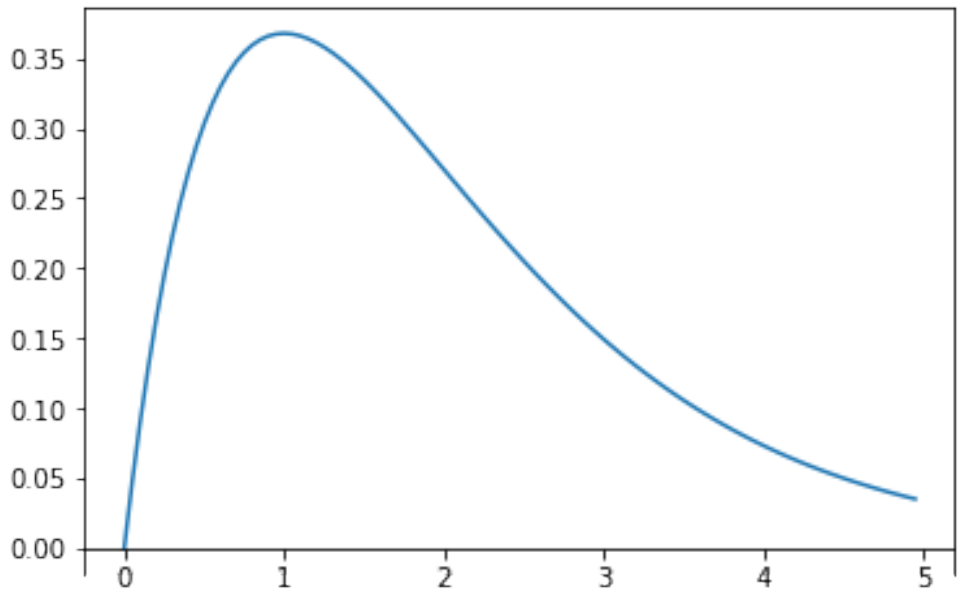
$$\left(a + \frac{i}{100}(b-a)\right)_{0 \leq i < 100},$$

ou de la notation ensembliste

$$\left\{a + \frac{i}{100}(b-a) \mid 0 \leq i < 100\right\}.$$

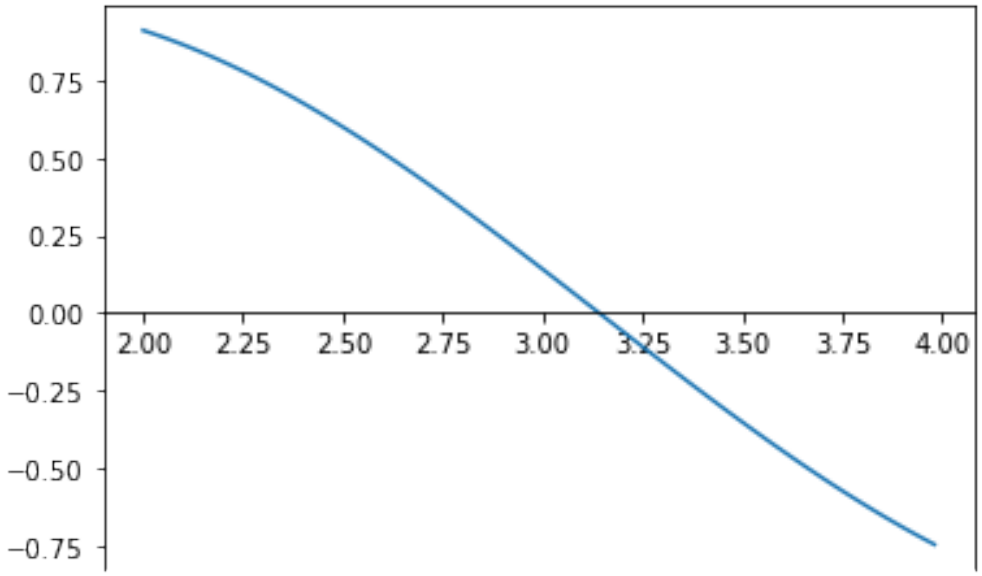
Remarque : on peut passer une fonction Python en paramètre d'une autre fonction. C'est très pratique. On peut aussi passer une fonction en paramètre sans l'avoir définie comme une fonction Python avec `lambda` : par exemple `lambda x : sin(x)` est la fonction qui prend en entrée x et retourne $\sin(x)$. Dans l'exemple suivant, on trace le graphe de la fonction $x \mapsto x.e^{-x}$.

```
In [4]: graphe(lambda x : x*exp(-x), 0,5)
```



Revenons à la fonction fsin définie ci-dessus. Cette fonction a manifestement plusieurs zéros dans l'intervalle $[-4, 4]$. On va s'intéresser au zéro se trouvant entre 2 et 4. On fait un zoom pour voir ce qui se passe.

```
In [5]: graphe(fsin, 2, 4)
```



Dans les sections A, B et C, nous allons voir trois façons d'obtenir une valeur approchée d'un zéro d'une fonction en la testant à chaque fois sur cet exemple, avant d'appliquer nos algorithmes à diverses fonctions dans la section D.

1.1.1 A. Méthode de dichotomie

Si on a une fonction réelle continue f et deux points a et b tels que $f(a)$ et $f(b)$ sont non nuls et de signes opposés, on sait par le théorème des valeurs intermédiaires que f s'annule dans l'intervalle $]a, b[$. Le principe de la dichotomie consiste à couper cet intervalle en deux en considérant le point milieu $m = (a + b)/2$:

- si $f(m) = 0$, on a trouvé un zéro de f dans l'intervalle $]a, b[$;
- si $f(a)$ et $f(m)$ sont non nuls et de signes opposés, f dans l'intervalle $]a, m[$: on continue la recherche dans cet intervalle deux fois plus petit ;
- sinon, $f(m)$ et $f(b)$ ne sont pas de même signe et f a un zéro dans l'intervalle $]m, b[$: on poursuit la recherche dans cet intervalle.

Voici une fonction qui reçoit en paramètre une fonction `f` et deux nombres `a` et `b` formant un encadrement d'un zéro de `f`, et renvoie un nouvel intervalle comme décrit ci-dessus.

```
In [ ]: def dichotomie_iter(f, a, b):
            m = (a+b)/2
            if f(m) == 0:
                return m, m
            if f(a) * f(m) < 0:
                return a, m
            else:
                return m, b
```

Pour chercher un bon encadrement du zéro recherché, on peut par exemple itérer la fonction précédente un nombre fixé de fois, comme dans la fonction `dichotomie_nbpas(f, a, b, n)` ci-dessous. (On décide d'afficher la valeur obtenue à chaque étape pour voir ce qui se passe : cette ligne peut être supprimée ou "commentée" (transformée en commentaire à l'aide du caractère `#`.)

```
In [ ]: def dichotomie_nbpas(f, a, b, n): # n pas
            for i in range(n):
                a, b = dichotomie_iter(f, a, b)
                print((a+b)/2)
            return (a+b)/2

In [ ]: dichotomie_nbpas(fsin,3,4,10)
```

Exercice

1. Écrire une fonction `dichotomie_eps(f, a, b, e)` qui prend en entrée une fonction f, deux nombres a et b encadrant un zéro de f, et $e > 0$, et qui renvoie une valeur approchée d'un zéro de f dans l'intervalle $]a, b[$ à e près.

```
In [ ]:
```

2. Tester sur la fonction $x \mapsto \sin x$ entre 3 et 4 pour obtenir une valeur approchée du zéro à 10^{-5} près.

```
In [ ]:
```

1.1.2 B. Méthode de Newton

On suppose maintenant que f est suffisamment régulière, et qu'on a accès à sa dérivée.

La méthode de Newton est une méthode itérative. On part d'un point x_0 assez proche du zéro qu'on veut approximer.

À chaque itération, on calcule un nouveau point x_{n+1} à partir du point précédent x_n. Le point x_{n+1} est défini de la manière suivante : c'est l'abscisse du point d'intersection de :

- la tangente T à la courbe de f en le point d'abscisse x_n,
- l'axe des abscisses.

La tangente T a pour équation $y = f(x_n) + (x - x_n) \cdot f'(x_n)$, ce qui permet de déterminer x_{n+1} à partir de x_n et des valeurs de f et f' en ce point :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Sous les bonnes hypothèses, la suite (x_n) converge vers le zéro de f proche de x_0.

On donne ci-dessous la fonction `newton_iter(f, df, x)` prenant la fonction, sa fonction dérivée et un point, et retourant le point obtenu en appliquant une itération comme décrit ci-dessus.

```
In [ ]: def newton_iter(f, df, x):
            return x - f(x) / df(x)
```

Exercice

1. Écrire deux fonctions :
   - `newton_nbpas(f, df, x, n)` qui retourne le point x_n obtenu après n itérations en partant de $x_0 = x$,
   - `newton_eps(f, df, x, e)` qui retourne le premier point x_n vérifiant $|x_n - x_{n-1}| < e$.

```
In [ ]:
```

2. Appliquer à la fonction sin en partant du point 4 : qu'obtient-on en 10 itérations? Combien d'itérations sont nécessaires pour obtenir $|x_n - x_{n-1}| < 10^{-10}$ en partant du point 4 ? Sur cet exemple, la convergence est-elle plus rapide ou plus lente que la méthode de dichotomie?

```
In [ ]:
```

1.1.3 C. Méthode de la sécante

Il se trouve des cas où on n'a pas accès à la dérivée de la fonction f dont on chercher à approcher un zéro. C'est le cas par exemple si f est calculée par un algorithme complexe. Il se peut aussi que f ne soit pas dérivable.

La méthode de la sécante est une méthode itérative. On part de deux points x_0 et x_1 proches du zéro de f à approcher.

Une itération permet de calculer un nouveau point x_{n+1} à partir des deux points précédents x_n et x_{n-1}. Le point x_{n+1} est l'abscisse du point d'intersection de

- la droite coupant la courbe représentative de f aux points d'abscisses x_n et x_{n-1},
- l'axe des abscisses.

Un calcul simple nous donne

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \cdot f(x_n).$$

Cette méthode est très similaire à celle de Newton : on a remplacé la tangente à la courbe de f à l'abscisse x_n par la droite coupant la courbe de f en les deux points rapprochés d'abscisses x_n et x_{n-1}.

Remarque : il n'est pas nécessaire que les deux points de départ x_0 et x_1 encadrent la racine (ils peuvent se situer du même côté).

Exercice

1. Écrire la fonction `secante_iter(f, x0, x1)` retournant l'abscisse `x_2` du point obtenu après une itération à partir des abscisses `x0` et `x1`.

```
In [ ]:
```

2. Écrire les fonctions `secante_nbpas(f, x0, x1, n)` et `secante_eps(f, x0, x1, e)` similaires à celles décrites pour la méthode de Newton.

```
In [ ]:
```

3. Tester ces fonctions sur la fonction $x \mapsto \sin x$, en partant par exemple des points $x_0 = 3.9$ et $x_1 = 4$.

```
In [ ]:
```

1.1.4 D. Quelques exemples de recherche de zéro

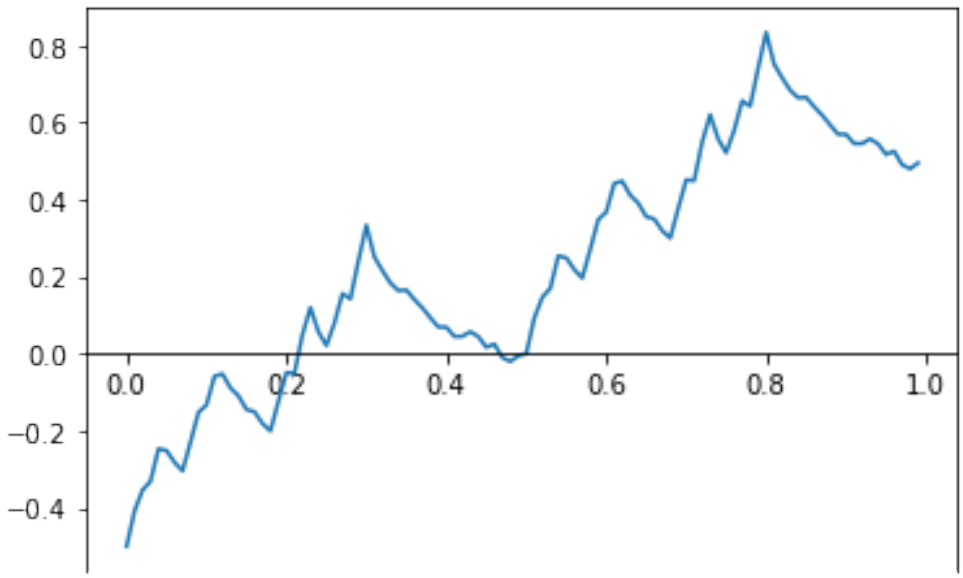
Exemple 1 : fonction polynomiale On considère le polynôme $P = X^5 - 3X^3 + 1$. Rechercher une valeur approchée de toutes les racines réelles de P à 10^{-9} près.

```
In [ ]:
```

Exemple 2 : la fonction de Bolzano C'est une fonction fractale définie de manière très similaire au flocon de Koch vue dans la feuille de géométrie. Elle est ici définie sur $[0,1]$. Par construction, elle est continue mais pas dérivable.

```
In [6]: def bolzano(x):
            xa, xb, ya, yb = 0, 1, -0.5, 0.5
            while xb - xa > 10**(-13):
                xl, yl = xb-xa, yb-ya
                if x <= xa + 3/8*xl:
                    xa, xb, ya, yb = xa, xa + 3/8*xl, ya, ya + 5/8*yl
                elif x <= xa + 1/2*xl:
                    xa, xb, ya, yb = xa + 3/8*xl, xa + 1/2*xl, ya + 5/8*yl, ya + 1/2*yl
                elif x <= xa + 7/8*xl:
                    xa, xb, ya, yb = xa + 1/2*xl, xa + 7/8*xl, ya + 1/2*yl, yb + 1/8*yl
                else:
                    xa, xb, ya, yb = xa + 7/8*xl, xb, yb + 1/8*yl, yb
            return (ya+yb)/2
```

```
In [7]: graphe(bolzano, 0, 1)
```



Par construction, la fonction de Bolzano a un zéro en $1/2$. Mais elle a (au moins) un autre zéro autour de 0.2. Utiliser une des méthodes vues ci-dessus pour approcher un zéro de cette fonction autour de 0.2.

```
In [ ]:
```

Exemple 3 : à propos du point de départ dans la méthode de Newton On considère la fonction f définie par $f(x) = 1/x + \ln(1+x) - 4$ pour $x > 0$, non définie sinon. C'est une fonction bien

régulière, on va utiliser la méthode de Newton pour approcher ses zéros. On commence par définir cette fonction ainsi que sa dérivée et on trace le graphe de f.

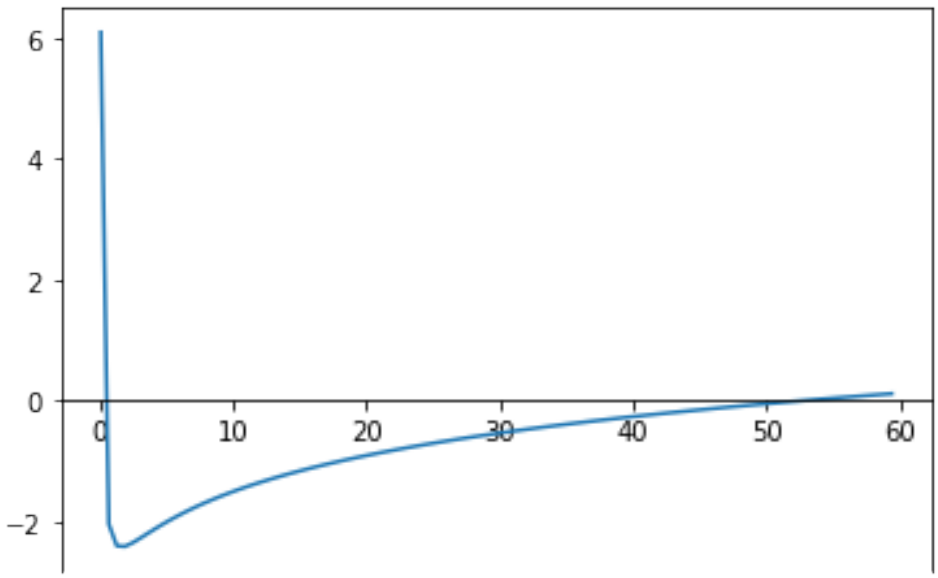
```
In [8]: def f(x):
            if x <= 0:
                # évaluer f en un point négatif ou nul provoque une erreur
                raise ValueError('fonction non définie')

            # le logarithme népérien s'écrit log en Python
            return 1/x+log(1+x)-4

        def df(x):
            if x <= 0:
                raise ValueError('fonction non définie')

            return -1/x**2+1/(1+x)

        graphe(f, 0.1, 60)
```



La fonction semble avoir un zéro proche de 0 et un autre autour de 50.

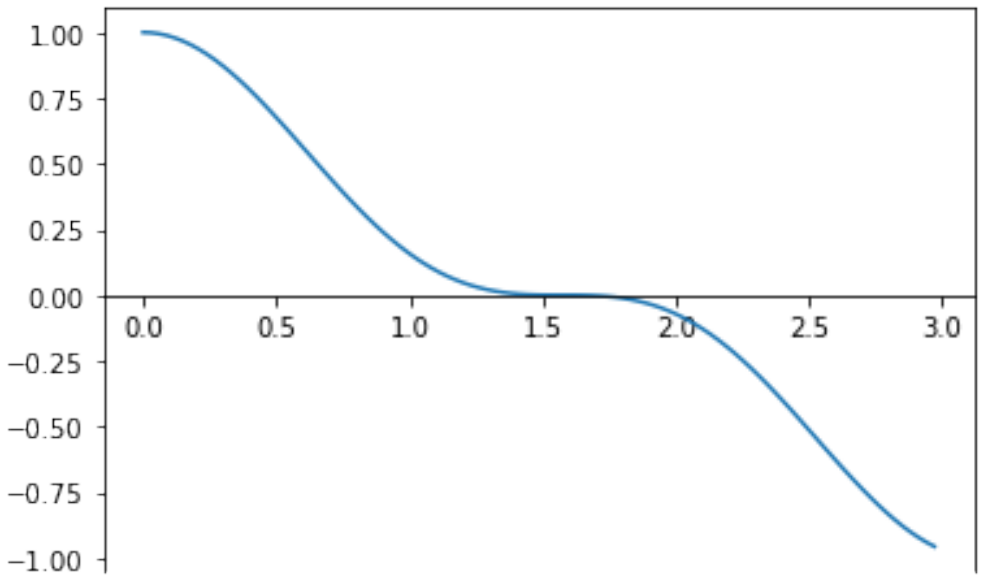
Que se passe-t-il quand on applique la méthode de Newton en partant de 2? En partant de 1? Trouver une valeur approchée du zéro proche de 0.

```
In [ ]:
```

Exemple 4 : à propos de la vitesse de convergence de la méthode de Newton On considère la fonction $x \mapsto \cos(x)^3$. On trace son graphe sur l'intervalle $[0, 3]$.

```
In [9]: def f(x):
            return cos(x)**3

        graphe(f, 0, 3)
```



Cette fonction a un zéro autour de $3/2$ (c'est bien sûr $\pi/2$). On va regarder la vitesse de convergence des méthodes de dichotomie et de Newton sur cette fonction.

1. Adapter la fonction `dichotomie_eps(f, a, b, e)` pour obtenir une nouvelle fonction `nbetapes_dichotomie_eps(f, a, b, e)` qui retourne non plus le point obtenu quand la taille de l'intervalle est inférieure à e mais le nombre d'étapes qui ont été nécessaires.

```
In [ ]:
```

2. Faire de même à partir de la fonction `newton_eps(f, df, x, e)` pour obtenir une nouvelle fonction `nbetapes_newton_eps(f, df, x, e)`.

```
In [ ]:
```

3. Pour $i \in \{0, 1, \ldots, 12\}$, calculer le nombre d'étapes nécessaires aux méthodes de Newton et de dichotomie pour la valeur $e = 10^{-i}$. Placer ces éléments dans deux listes et tracer le graphique du nombre d'étapes en fonction de i. Que remarque-t-on?

```
In [ ]:
```

1.1.5 Remarques et références

La méthode de dichotomie a une convergence linéaire au sens où on obtient de l'ordre de n décimales après n itérations (attention, on obtient en réalité n décimales correctes en base 2 uniquement, moins en base 10 ! Mais la précision est bien de l'ordre de $1/10^{c.n}$ pour une constante c).

Sous de bonnes hypothèses, la méthode de Newton a une convergence quadratique : on obtient de l'ordre de n^2 décimales après n itérations. Pour l'étude précise des hypothèses permettant de garantir la convergence des méthodes de Newton ou de la sécante et l'analyse de la vitesse de convergence de ces méthodes, consulter Demailly.

A propos de l'exemple de la fonction de Bolzano : une fonction continue f peut être très compliquée, il est difficile de garantir le comportement d'un algorithme tel que celui de la sécante sur une telle fonction. Mais il est bien sûr possible d'appliquer l'algorithme et de voir ce qui est obtenu. Si on demande à approcher un zéro à ε près et qu'on obtient un point x, on peut tester si $f(x-\varepsilon)\cdot f(x+\varepsilon) < 0$. Si c'est le cas, on sait que la fonction f change de signe et possède donc un zéro dans l'intervalle $[x-\varepsilon, x+\varepsilon]$. La définition précise de la fonction de Bolzano est disponible sur Wikipedia (la version qu'on a définie est décalée de $1/2$ vers le bas).

A propos de la recherche des zéros d'un polynôme : il existe des algorithmes exacts très efficaces pour compter le nombre de racines réelles d'un polynôme dans un intervalle (dont les extrémités sont finies ou infinies), et isoler les racines, c'est-à-dire trouver pour chaque racine un intervalle contenant cette racine et uniquement celle-ci. La phase d'analyse globale du polynôme pour déterminer son nombre de racines est donc complètement automatisée. Pour plus de détails sur ce sujet consulter le livre de Basu, Pollack et Roy.

- Jean-Pierre Demailly. Analyse numérique et équations différentielles. Presses universitaires de Grenoble, 1996.
- Fonction de Bolzano, Wikipedia. https://fr.wikipedia.org/wiki/Fonction_de_Bolzano
- Basu, Pollack, Roy. Algorithms in real algebraic geometry. Springer, 2006.

Remarques générales :

1) Pourquoi ne pas avoir mis les schémas juste après les avoir mentionnés ?