Université de Montréal

La programmation informatique dans la recherche et la formation en mathématiques au niveau universitaire

par
Laura Broley

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.) en mathématiques
option mathématiques appliquées

juillet, 2015

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé

La programmation informatique dans la recherche et la formation en mathématiques au niveau universitaire

présenté par
Laura Broley

a été évalué par un jury composé des personnes suivantes :

Sabin Lessard
(président-rapporteur)

Yvan Saint-Aubin
(directeur de recherche)

France Caron
(directrice de recherche)

Jacques Bélair
(membre du jury)

Mémoire accepté le
19 août 2015

Résumé

Une étude récente auprès de 302 mathématiciens canadiens révèle un écart intriguant : tandis que 43% des sondés utilisent la programmation informatique dans leur recherche, seulement 18% indiquent qu'ils emploient cette technologie dans leur enseignement (Buteau et coll., 2014). La première donnée reflète le potentiel énorme qu'a la programmation pour faire et apprendre des mathématiques. La deuxième donnée a inspiré ce mémoire : pourquoi existe-t-il un tel écart ? Pour répondre à cette question, nous avons mené une étude exploratoire qui cherche à mieux comprendre la place de la programmation dans la recherche et la formation en mathématiques au niveau universitaire. Des entrevues semi-dirigées ont été conduites avec 14 mathématiciens travaillant dans des domaines variés et à différentes universités à travers le pays. Notre analyse qualitative nous permet de décrire les façons dont ces mathématiciens construisent des programmes informatiques afin d'accomplir plusieurs tâches (p.e., simuler des phénomènes réels, faire des mathématiques « expérimentales », développer de nouveaux outils puissants). Elle nous permet également d'identifier des moments où les mathématiciens exposent leurs étudiants à certains éléments de ces pratiques en recherche. Nous notons toutefois que les étudiants sont rarement invités à concevoir et à écrire leurs propres programmes. Enfin, nos participants évoquent plusieurs contraintes institutionnelles : le curriculum, la culture départementale, les ressources humaines, les traditions en mathématiques, etc. Quelques-unes de ces contraintes, qui semblent limiter l'expérience mathématique des étudiants de premier cycle, pourraient être revues.

Mots-clés : pratiques mathématiques, enseignement universitaire, programmation informatique, curriculum mathématique, contraintes institutionnelles

Abstract

A recent survey of 302 Canadian mathematicians points to an intriguing gap: while 43% of the participants use computer programming in their research, only 18% indicate that they use such technology in their teaching (Buteau et al., 2014). The first statistic reflects the enormous potential that programming has for doing and learning mathematics. The second served as the inspiration for our research: why would such a gap exist? In response to this question, we put forth an exploratory study aimed at better understanding the place of programming in mathematical research and university mathematics education. Semi-directed interviews were conducted with 14 mathematicians working within various mathematical subfields at different universities across Canada. Our qualitative analysis allows us to describe the ways in which these mathematicians construct computer programs in order to accomplish several tasks (e.g., simulating real-world phenomena, doing "experimental" mathematics, developing new powerful tools). It also allows us to identify some moments where the mathematicians expose their students to certain elements of these research practices. We notice, however, that the students are rarely invited to conceptualize and write their own programs. In the end, our participants highlight several institutional constraints: the curriculum, departmental culture, human resources, the traditions in mathematics, etc. Some of these constraints, which seem to be limiting the mathematical experience of some undergraduate students, could warrant re-examination.

Keywords : mathematical practices, university teaching, computer programming, mathematics curriculum, institutional constraints

Table des matières

Liste des figures

Ceci, comme tout, est pour Mitch et Kim.

Remerciements

It is hard to believe that two years have passed since I began my journey at Université de Montréal; and what a journey it has been! Like any other incredible opportunity for learning and growth, it has been accompanied by several challenges and successes. There have been some moments of doubt and discouragement; however, I finish with the feeling that it has truly been an amazing experience. In the end, this paper symbolizes much more than an academic achievement: it celebrates a beautiful period of transition, hard work, and a whole lot of self-discovery. But the great strides I have made, in all interconnecting aspects of my life, is in large part due to the support I have received from so many different people.

It is impossible to express in words the thanks I owe to the two people who have been by my side every step of the way: Mitchell, my partner in love and in life, and Kim, my mom and best friend. Thank you for listening. Thank you for understanding. Thank you for believing in me like no one else does. Thank you for your unconditional love, without which I don't know where I'd be. This is as much for you as it is for me.

Deux autres personnes méritent un énorme merci : France Caron et Yvan Saint-Aubin, qui ont été beaucoup plus que mes directeurs de recherche pendant ces deux années. Ce fut un grand honneur de travailler avec vous deux et d'apprendre de vos expertises. Je ne peux pas vous remercier assez pour le soutien que vous m'avez fourni, autant dans ma vie académique que dans les défis personnels que j'ai rencontrés. Sous votre direction, j'ai grandi plus que je ne croyais possible, comme mathématicienne, comme chercheuse et comme personne.

To all my family and friends, of which I am blessed to have so many, thank you for your prayers, thoughts, and other methods of support throughout this whole process. Jim, Brian, Mike, Dan, and Luke: thank you for always being there. Kristina, Krissa, Nicole, Katherine, and Helen: thank you for keeping me grounded. Cheryl and Maude: thank you for making Montréal my home. And Cyclone: thank you for your constant company.

I would like to say a very special thanks to Chantal Buteau, who accepted to supervise me three years ago as an undergraduate student, and who has been providing guidance ever since. Thank you for initiating me to the world of research. Thank you for providing me with

so many amazing opportunities over the years. Thank you for believing in me and pushing me. Without you, I wouldn't be here.

Getting here also would not have been possible without a few other very important people. Neal: thank you for sharing your experiences with me, opening my eyes to new prospects, and believing in my capabilities as a researcher. Eric: thank you for allowing me to take part in your Brock Bugs vision. A big thank you as well to the Department of Mathematics & Statistics at Brock University and the Brock Leaders Citizenship Society, both of which have taught me so much.

Au département de mathématiques et de statistique de l'Université de Montréal, qui a cru en moi et en mes idées : merci de votre ouverture à l'innovation qui m'a permis de bénéficier d'une formation exceptionnelle.

Merci aussi au gouvernement du Canada, au Conseil de recherches en sciences humaines, à la Faculté des arts et des sciences, à la Faculté des études supérieures et postdoctorales et à la Banque Nationale du Canada, qui m'ont donné le soutien financier nécessaire pour poursuivre mes intérêts, mes talents et mes rêves.

Finalement, j'aimerais remercier tous les mathématiciens qui partagèrent leurs expériences et leurs perspectives avec moi pendant mon étude ; sans leur participation, ce mémoire n'aurait pas été possible.

Introduction

C'est à l'université que j'ai découvert la programmation informatique comme une activité puissante pour faire et apprendre des mathématiques. Avant cette découverte, mon apprentissage des mathématiques s'était limité à des expériences plutôt passives en classe, où j'écoutais l'enseignant parler des concepts au tableau, et à la complétion de séries d'exercices pour perfectionner les techniques dont on évaluerait la maîtrise aux examens. Si les mathématiques présentées de cette façon demeuraient intéressantes, elles semblaient relever d'une discipline limitée et figée. Cependant, après avoir appris à développer des programmes pour explorer des idées mathématiques et résoudre des problèmes réels, ma vision a changé radicalement : les mathématiques me sont apparues alors comme une discipline beaucoup plus vaste, en constante évolution et à laquelle il me devenait possible de contribuer activement. Cette transformation enrichissante m'a conduite à être un peu perplexe face à l'intégration restreinte de la programmation à travers mon propre baccalauréat. Je me suis demandé : est-ce que la programmation aurait pu, ou aurait dû, être plus présente ?

Ainsi a commencé mon intérêt pour cerner la place et le rôle de la programmation dans l'enseignement des mathématiques au niveau universitaire et, plus généralement, de l'utilisation de programmes informatiques par les mathématiciens. Une première tentative de réponse m'a menée à poser les bases de la problématique à la source de ce projet. Au Chapitre 1, nous élaborons cette problématique afin d'identifier les objectifs généraux de notre recherche. Le Chapitre 2 précise les notions et théories qui nous aideront à déterminer des questions plus spécifiques. L'étude qualitative exploratoire que nous avons effectuée pour répondre à ces questions est décrite au Chapitre 3, et les résultats principaux sont dévoilés et discutés aux Chapitres 4 et 5. Le Chapitre 6 présente enfin les conclusions du mémoire.

1 Problématique

Dans ce premier chapitre, nous explorons le contexte de la programmation dans la recherche et la formation en mathématiques au niveau universitaire. Cette exploration nous mène à notre problème de recherche et nos objectifs généraux.

1.1 La programmation en recherche : importante, mais cachée

Personne ne peut nier que l'invention et l'évolution subséquente de l'ordinateur aient apporté des changements radicaux dans pratiquement tous les domaines de l'activité humaine, y compris celui où œuvrent les mathématiciens. En 1992, Cornu résumait les principaux apports de cet outil : on peut calculer (mieux, plus vite, sur un plus grand domaine de nombres), visualiser (de nouveaux phénomènes, régularités), expérimenter (faire des observations sur plusieurs exemples, arriver à des hypothèses, vérifier des conjectures), simuler (des phénomènes trop coûteux, trop longs à réaliser sans ordinateur) et démontrer (soit par une « démonstration automatique », soit en utilisant des contributions informatiques). Ces changements tangibles apportés par l'ordinateur à l'activité mathématique ont même favorisé le développement de nouveaux domaines (p.e., les méthodes de calcul numérique) et de nouvelles applications (p.e., dans la cryptographie, les fractales, la compression d'images).

Mais l'utilisation de l'ordinateur peut aujourd'hui s'appuyer sur des logiciels puissants ; que peut-on dire alors de la programmation faite par les mathématiciens ? Dans une étude récente auprès de 302 mathématiciens canadiens, Buteau, Jarvis et Lavicza (2014) ont trouvé que 43% des sondés programment dans leur recherche ; il s'agit même du deuxième mode d'utilisation des ordinateurs le plus prisé par les mathématiciens (après les logiciels de calcul symbolique, qui ont reçu un taux d'utilisation de 65%). Si l'on peut en inférer que la programmation offre un certain potentiel pour la recherche, l'étude, dans sa méthodologie quantitative, ne renseigne pas sur les façons exactes dont les mathématiciens développent et utilisent des programmes informatiques. En fait, nous n'avons trouvé aucune étude qui présente une discussion plus approfondie de l'intégration de la programmation dans le travail de recherche des mathématiciens. Cette partie de la recherche mathématique demeure encore méconnue, et cela nous paraît justifier son exploration.

1.2 La programmation en enseignement : un potentiel énorme

Parallèlement à l'intégration de l'ordinateur dans l'activité mathématique, les didacticiens se sont intéressés à l'utilisation de cet outil pour enrichir l'enseignement et l'apprentissage des mathématiques. En effet, la première étude de l'*International Commission on Mathematical Instruction*, parue en 1986, était consacrée à l'influence des ordinateurs et de l'informatique sur les mathématiques elles-mêmes et sur leur enseignement (Churchhouse et coll., cité dans Buteau et coll., 2014). L'activité de programmation en particulier a longtemps été reconnue comme apportant un potentiel énorme à l'instruction mathématique.

Feurzeig et Papert étaient parmi les pionniers à avoir fortement encouragé la mise en contact des élèves en mathématiques avec la programmation. Pendant les années soixante, ils ont développé un langage de programmation, *LOGO*, dans le but de rendre la programmation et ses avantages plus accessibles aux jeunes élèves. Un rapport écrit par ces chercheurs en 1968 (voir Feurzeig et Papert, 2011) décrit leur vision initiale. Ils suggèrent tout d'abord que des activités de programmation peuvent aider les élèves à développer une nouvelle approche expérimentale pour résoudre des problèmes mathématiques, telle que celle employée par des mathématiciens dans leurs recherches. Éventuellement, un langage de programmation peut même devenir un instrument puissant dans leur boîte à outils, un instrument qui leur offre la possibilité d'ouvrir de façon importante l'ensemble des tâches qu'ils sont capables d'accomplir, qu'elles soient mathématiques ou non.

Feurzeig et Papert (2011) mettent aussi en évidence plusieurs avantages qui débordent l'appropriation des méthodes informatiques utilisées par les vrais mathématiciens. Ils expliquent, par exemple, que certains aspects de la programmation sont semblables à la résolution de problèmes mathématiques : dans les deux cas, on identifie des sous-problèmes, formule un plan, vérifie des réponses, corrige des erreurs, etc. L'étudiant qui programme peut donc développer des compétences qui peuvent lui être utiles dans d'autres contextes mathématiques. Des activités de programmation peuvent également fournir aux étudiants un langage et un ensemble d'expériences utiles pour discuter des mathématiques avec d'autres et pour mieux comprendre leurs propres façons de penser. En effet, l'apprentissage et l'utilisation d'un langage de programmation peuvent faciliter l'enseignement de certains concepts

mathématiques associables à des concepts semblables en informatique (p.e., les concepts de variable, de fonction, de récursivité). Enfin, pour être en mesure de construire les programmes qu'ils veulent construire, les étudiants peuvent avoir besoin d'apprendre et d'employer des notions qui leur apparaissaient abstraites ; l'application de ces notions dans le contexte de programmation peut les rendre plus concrètes et plus faciles à apprécier par conséquent.

Si le travail de Feurzeig et de Papert a principalement ciblé l'intégration de la programmation dans l'apprentissage des mathématiques chez de jeunes élèves, on peut se demander si les étudiants d'un premier cycle[1] en mathématiques pourraient vivre des effets positifs semblables, à leur niveau. Cette question a été examinée dans plusieurs études, qui proposent même des avantages additionnels pour ces étudiants. Quelques chercheurs (p.e., Leron et Dubinsky, 1995 ; Wilensky, 1995) et praticiens (p.e., Pesonen et Malvela, 2000) affirment que la programmation peut contribuer à la compréhension de concepts complexes qui ne sont pas directement liés aux idées provenant de l'informatique. Leron et Dubinsky (1995), par exemple, décrivent une nouvelle façon d'enseigner les notions d'algèbre abstraite qui implique l'écriture de programmes informatiques par les étudiants. Leur méthode s'appuie sur l'hypothèse que la programmation de concepts mathématiques peut exiger une certaine construction mentale de ces concepts. En utilisant leurs programmes pour explorer plusieurs exemples, les étudiants peuvent être ensuite amenés à affiner progressivement leur compréhension. Et à l'aide de la base concrète développée pendant ces explorations, ils peuvent être mieux préparés à une version plus abstraite des concepts, par exemple, sous la forme d'une définition formelle. L'expérience d'implémentation de l'approche ainsi qu'une première validation rigoureuse de son impact sur la compréhension des notions spécifiques (voir Asiala, Dubinsky, Mathews, Morics et Oktaç, 1997) présentent des résultats prometteurs.

D'autres chercheurs avancent que les étudiants de premier cycle qui écrivent et utilisent des programmes peuvent développer des compétences précieuses en mathématiques. Après

[1] Au Québec, on utilise les mots « premier cycle » pour décrire l'ensemble des études universitaires qui inclut typiquement trois ans de scolarité commençant à la 14^{e} année, après les études primaires, secondaires et collégiales. Dans les autres provinces du Canada, un premier cycle en mathématiques peut durer quatre ans, commençant à la 13^{e} année.

avoir conduit une vaste revue de la littérature, Marshall (2012) donne un résumé de telles habiletés selon les étapes différentes dans la construction d'un programme informatique. Durant une phase de préparation, l'étudiant peut développer son intuition mathématique, sa compréhension des modèles mathématiques et son aptitude à réfléchir face aux problèmes. Pendant l'écriture d'un programme, l'étudiant peut apprendre non seulement à programmer des mathématiques, avec toute la rigueur qu'impose le respect des structures et du langage informatique, mais aussi à travailler avec des objets abstraits et à juger de la justesse de son travail. En utilisant ses programmes, un étudiant peut améliorer ses compétences pour interpréter et travailler avec plusieurs représentations de résultats mathématiques. Et en général, l'étudiant qui participe à des activités de programmation peut apprendre à faire des mathématiques d'une façon autonome et à s'engager dans des projets de recherche.

Finalement, pour les étudiants au baccalauréat, le potentiel de la programmation s'étend au-delà de l'individu et de l'immédiat. Comme j'ai pu l'apprécier durant mon premier cycle, l'occasion de programmer contribue à une culture d'apprentissage où l'étudiant se voit comme participant actif dans un monde mathématique vivant (Muller, Buteau, Ralph et Mgombelo, 2008 ; Wilensky, 1995). De plus, la programmation offre la possibilité d'élargir considérablement le champ des problèmes accessibles aux étudiants dans la mesure où ils sont exposés aux nouvelles approches de résolution et aux nouveaux concepts rendus possibles, accessibles, visibles, manipulables ou même nécessaires par les outils informatiques. Le développement de compétences de programmation peut ainsi diversifier les débouchés des étudiants. Après tout, un rapport fait en 2012 indique que pour les mathématiciens qui travaillent en industrie, « programming and computer skills continue to be the most important technical skill that new hires bring to their jobs » (Society for Industrial and Applied Mathematics, p. 25). Et les résultats de Buteau et coll. (2014) suggèrent que de telles compétences peuvent être également importantes pour les étudiants qui visent une carrière de mathématicien dans le milieu académique.

Ainsi plusieurs mathématiciens et didacticiens reconnaissent un potentiel énorme à l'intégration d'activités de programmation dans l'enseignement des mathématiques au niveau universitaire ; mais ils ne documentent que rarement la manière dont la programmation est intégrée à la formation. Presque tous les articles que nous avons lus ont une nature théorique

ou décrivent des expériences uniques qui se passent en dehors des cours (p.e., Abrahamson, Berland, Shapiro, Unterman et Wilensky, 2004 ; Wilensky, 1995) ou pendant des cours spécifiques à des universités particulières (p.e., Leron et Dubinksy, 1995 ; Muller et coll., 2008 ; Pesonen et Malvela, 2000). Que peut-on dire alors de l'intégration plus générale de la programmation dans les programmes de premier cycle en mathématiques ?

1.3 Un écart notable et intriguant

Un élément de réponse à cette question nous vient de l'étude de Buteau et coll. (2014) concernant l'utilisation d'outils informatiques par des mathématiciens canadiens. La Figure 1.1 présente un diagramme de cette étude.

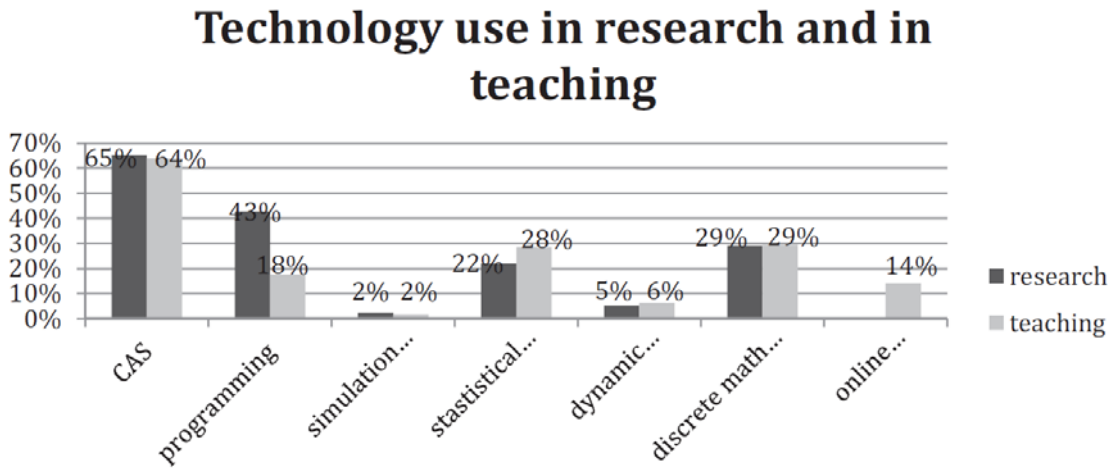


Figure 1.1 Utilisation de la technologie dans la recherche et dans l'enseignement (diagramme extrait de Buteau et coll., 2014).

On y remarque que, pour presque tous les outils considérés, l'utilisation en recherche et l'utilisation en enseignement sont presque identiques. Ce n'est pas le cas cependant pour l'utilisation de la programmation. Tandis que 43% des mathématiciens ayant répondu au sondage écrivent des programmes informatiques dans leur recherche, seulement 18% intègrent la programmation dans leur enseignement. Pourquoi existe-t-il un tel écart ?

Buteau et ses collaborateurs (2014) suggèrent que la courbe d'apprentissage est beaucoup plus grande pour la programmation que pour la simple utilisation d'autres outils informatiques qui ont des interfaces plus conviviales. Ils mentionnent aussi la nécessité de surmonter certains obstacles logistiques et systématiques : un département de mathématiques

qui veut intégrer la programmation doit déterminer comment l'enseigner et comment l'incorporer dans d'autres cours. Si nous faisons l'hypothèse que les difficultés d'intégrer les logiciels de calcul symbolique sont aussi applicables à l'intégration de la programmation, d'autres contraintes pourraient s'ajouter aux explications fournies par Buteau et coll. (2014). Par exemple, parmi les sources de difficultés relevées par Marshall, Buteau, Jarvis et Lavicza (2012), on note l'introduction d'outils informatiques dans les évaluations d'un cours, la détermination d'un bon équilibre entre l'incorporation de l'ordinateur et l'emploi des techniques à la main, l'utilisation de l'ordinateur pour atteindre des buts pédagogiques (et pas seulement pour s'amuser) et le temps pour développer et intégrer des activités appropriées. Ceci étant dit, nous n'avons trouvé aucune étude qui ait cherché à vérifier de telles hypothèses ou à explorer la possibilité d'autres explications pour l'écart notable et intriguant observé en Figure 1.1. Face à ce vide à combler, nous énonçons nos objectifs généraux de recherche.

1.4 Nos objectifs généraux

Le présent projet de recherche cherche à mieux comprendre :

1) la place, le rôle, le potentiel et les contraintes de la programmation dans la recherche mathématique ;
2) la place, le rôle, le potentiel et les contraintes de la programmation dans l'enseignement des mathématiques au niveau universitaire ;
3) les liens et les écarts entre l'intégration de la programmation par des mathématiciens dans leur recherche et dans leur enseignement.

En apportant un éclairage à une partie relativement cachée de la recherche des mathématiciens et en examinant leurs méthodes d'enseignement liées à la programmation, notre projet pourrait conduire à de nouvelles idées au regard de l'intégration d'outils informatiques dans l'enseignement des mathématiques. Après tout, il y a relativement peu de recherches récentes au sujet de l'utilisation de l'ordinateur dans les mathématiques de premier cycle (Lavicza, 2010). À long terme, nos conclusions pourraient connaître des retombées dans le développement des programmes de formation mathématique universitaires et pré-universitaires.

2 Cadre théorique

Dans ce chapitre, nous précisons l'ensemble d'outils théoriques qui nous aidera à aborder la problématique identifiée au Chapitre 1. Nous commençons en développant une définition de base de la programmation. Nous introduisons ensuite quelques notions qui nous permettront de décrire et d'analyser la place de la programmation dans la recherche et dans l'enseignement des mathématiciens. Nous terminons le chapitre avec les questions spécifiques ciblées par notre projet.

2.1 La programmation : une définition à clarifier

Étant donné l'importance évidente du mot « programmation » pour notre étude, il nous apparaît essentiel d'en élaborer une définition. Le Nouveau Petit Robert (Robert, 2010) définit la programmation comme l'élaboration et la codification d'un programme, lequel serait un « ensemble des instructions, rédigé dans un langage de programmation, permettant à un système informatique d'exécuter une tâche donnée » (p. 2038). Si cette définition fournit un bon point de départ, elle ne répond pas à toutes les questions qui se posent quand nous essayons de tracer la frontière entre ce qui constitue la programmation et ce qui n'en est pas.

Par exemple, si la réalisation d'un projet repose sur l'implémentation d'un algorithme (c.-à-d., un ensemble d'instructions) pour exécuter une tâche donnée, un chercheur pourrait devoir ou vouloir coder chaque partie de l'algorithme lui-même, ce qui pourrait le mener à écrire plusieurs centaines de lignes de code. Mais il se peut que l'accomplissement de la même tâche soit possible en beaucoup moins de lignes si le chercheur fait appel aux fonctions préprogrammées dans un logiciel plus évolué. Peut-être le chercheur a-t-il même accès à un programme informatique complet, développé par quelqu'un d'autre, qui est capable de compléter sa tâche avec peu ou pas de changements dans le code. Dans quels cas le travail du chercheur peut-il être considéré comme de « la programmation » ?

Les travaux que nous avons lus ne suggèrent pas de réponse claire à cette question. Buteau et coll. (2014), par exemple, identifient la programmation à l'emploi des langages comme *Java, C++* ou *Fortran* ; ils renvoient l'utilisation de *Maple* ou de *Mathematica* à une

première catégorie distincte (celle des logiciels de calcul symbolique) et l'utilisation de *MATLAB* ou d'*Excel* à une seconde catégorie distincte (celle des systèmes de mathématiques discrètes). Des différences marquées entre ces classes d'outils justifient leur distinction. Nous devons noter pourtant que la construction de programmes est tout à fait possible avec les logiciels *Maple, Mathematica, MATLAB* ou *Excel*, au même titre qu'avec des langages dédiés à la programmation (*Fortran*, *Java*, *C++*, etc.).

La décomposition en étapes de l'activité de programmation nous paraît encore plus complexe que d'en donner une définition. À cet égard, la littérature offre plusieurs idées, mais une structuration semble encore à faire. Dans leur description du processus vécu par leurs étudiants en mathématiques qui créent des environnements informatiques pour explorer des conjectures abstraites ou des applications réelles, Buteau et Muller (2010) parlent d'un cycle de programmation qui inclut *le design*, *l'implantation*, *les tests* et *les révisions* de l'environnement. Marshall (2012) indique que l'exploration mathématique effectuée avec ces environnements peut résulter de la programmation, mais n'en fait pas vraiment partie. Similairement, l'étape où l'on définit un problème à explorer peut entraîner un besoin de programmer, mais elle ne doit pas être considérée comme faisant partie intégrante de l'activité de programmation. Par contre, Marshall (2012) suggère que *l'analyse du problème* identifié peut être un élément important du cycle de programmation puisqu'elle peut influencer les choix de design. Cette idée est aussi présente chez Feurzeig et Papert (2011) qui proposent de considérer la programmation comme un cas particulier de la résolution de problèmes développée par Polya. En 1945, ce mathématicien avait défini le processus de résolution de problèmes comme étant constitué de quatre sous-processus : l'analyse du problème, la définition d'un plan, l'exécution du plan et un retour réflexif sur la solution (cité dans Caron, 2001). En poussant plus loin cette analogie, nous trouvons une autre étape possible : une phase de *contrôle* responsable de diriger les transitions (en avant ou en arrière) entre deux sous-processus (Schoenfeld, 1985).

Toutes les idées juste présentées ont été développées dans le contexte de l'enseignement des mathématiques. La considération des cycles de développement de logiciels utilisés par des informaticiens-mathématiciens travaillant en industrie (voir, p.e., Guntamukkala, Wen et Tarn, 2006) soulève cependant des idées plus complexes telles que la

spécification des buts d'un programme en fonction des outils informatiques disponibles, l'évaluation de plusieurs designs (incluant plusieurs architectures ou plusieurs algorithmes), la construction modulaire des programmes, l'adaptation à d'autres besoins (ceux, p.e., de nouveaux clients) et même la mise à jour des programmes au fil des ans. Il nous faudra voir si ces descriptions de la programmation s'appliquent au contexte de la recherche mathématique et si elles sont suffisantes pour la décrire.

Nous faisons donc le choix conscient de partir d'une définition plutôt large de la programmation, quitte à y ajouter, en cours d'analyse, des éléments susceptibles d'enrichir ou de raffiner cette première vision. En effet, le développement d'une définition explicite de la programmation sera le but de la section 4.1. Pour le moment, nous considérerons comme programmation toute activité qui cherche à contrôler le fonctionnement d'un ordinateur selon des buts précis et qui inclut un sous-ensemble d'étapes comme celles mentionnées ci-avant.

2.2 La complexité et l'utilité d'apprendre à programmer

Le concept de *genèse instrumentale* (Vérillon et Rabardel, 1995, cité dans Artigue, 2002) a déjà été appliqué dans plusieurs études concernant l'intégration d'outils informatiques dans l'enseignement des mathématiques pour rendre compte de la complexité et du caractère progressif de l'appropriation de ces outils. Nous l'introduisons dans notre projet pour expliquer à la fois la complexité et l'utilité d'apprendre à programmer.

Selon cette théorie, un individu face à un nouvel outil le voit comme un simple *artefact* : il ne sait pas a priori comment l'utiliser. Mais, après avoir associé à cet outil certains schèmes d'utilisation, l'artefact devient un *instrument*. Trouche (2000) explique que le processus de transformer un artefact en un instrument (la genèse instrumentale) se déroule à deux niveaux, souvent d'une manière inégale. D'un côté, le sujet pourrait choisir de s'adapter aux capacités de l'artefact (*l'instrumentation*) ; de l'autre côté, le sujet pourrait décider d'adapter l'artefact à ses propres besoins (*l'instrumentalisation*). À titre d'exemple, considérons le logiciel de calcul symbolique *Mathematica* comme artefact de départ. Ce logiciel intègre des fonctions puissantes dont l'apprentissage peut en faire un instrument très utile en mathématiques. Face au problème d'étudier le comportement des solutions numériques d'un système d'équations différentielles, on pourrait se contenter d'apprendre à utiliser les fonctions

« NDSolve » et « Plot » dans leurs formes les plus simples et de viser ainsi une instrumentation de base. Mais si le réglage par défaut ne produit pas des résultats adéquats pour faire les observations nécessaires, on pourrait décider de s'engager dans une instrumentalisation, en utilisant, par exemple, des options de « Plot » telles que la spécification des limites de l'affichage des axes, la détermination de la couleur des solutions, etc. On pourrait même aller plus loin dans l'instrumentalisation en renonçant à « NDSolve » et en écrivant une sous-routine plus appropriée pour résoudre le système à l'étude.

Évidemment, la genèse instrumentale varie selon la personne, l'outil avec lequel elle interagit et le contexte à l'intérieur duquel cette interaction se produit : elle dépend des habitudes et des connaissances de l'utilisateur, des potentialités et des contraintes induites par l'artefact et des caractéristiques de la situation d'utilisation spécifique (Trouche, 2000). De plus, la genèse instrumentale n'est ni automatique, ni immédiate, ni définitive ; elle peut demander beaucoup de temps et d'effort, s'effectuer à travers plusieurs étapes et continuer à évoluer au fil des ans (Artigue, 2002).

La complexité d'apprendre à programmer devient évidente alors lorsque nous considérons la complexité-même de l'artefact lié à cette activité. Un programme informatique se déploie typiquement sur plusieurs niveaux : par exemple, tout ce qui est à l'extérieur de l'ordinateur (clavier, souris, écran, etc.), l'interface (menus, zones d'input, graphiques, etc.), le code source (ensemble d'instructions codées dans un interpréteur), le langage de programmation (fonctions préprogrammées, règles fondamentales), le système d'exploitation (ensemble des programmes de bas niveau qui contrôlent l'utilisation des capacités d'un ordinateur par des logiciels d'applicatifs) et les composantes physiques de l'ordinateur (registres de mémoire, processeurs, etc.). Apprendre à programmer exige une certaine genèse instrumentale de plusieurs de ces éléments, qui apportent leurs propres contraintes et potentialités. Mais ce faisant, on n'est plus restreint à s'adapter au fonctionnement donné d'un ordinateur ou des logiciels existants ; les possibilités d'instrumentalisation deviennent nombreuses, d'où l'utilité de développer des compétences de programmation.

Pour progresser dans sa genèse instrumentale, on gagne à mieux comprendre les contraintes induites par l'outil (Artigue, 2002). Quand il s'agit d'un outil informatique utilisé pour faire des mathématiques, les contraintes sont souvent liées à la *transposition*

informatique des savoirs (Balacheff, 1994, cité dans Trouche, 2000), c'est-à-dire, la projection des mathématiques dans l'environnement informatique spécifique. Par exemple, la nature physique de l'ordinateur impose la discrétisation des phénomènes continus et, par conséquent, une image est toujours présentée sur l'écran par un nombre fini de pixels. Trouche (2000) distingue trois types de contraintes :

1) les *contraintes internes*, liées intrinsèquement à l'ordinateur (p.e., la limitation de la mémoire, la coordination entre les différents modes de calcul) ;
2) les *contraintes de commandes*, liées à la syntaxe et aux fonctions disponibles ;
3) les *contraintes d'organisation*, liées à l'organisation du clavier et de l'interface.

Ces contraintes influencent la programmation des mathématiciens et celle de leurs étudiants. Mais pour les étudiants, qui sont souvent des programmeurs débutants, les contraintes peuvent aussi avoir un impact énorme sur le développement de leurs compétences de programmation.

Selon Trouche (2004), la direction d'un professeur est essentielle pour gérer les contraintes et pour diriger la genèse instrumentale de l'étudiant. Il s'est intéressé donc aux *orchestrations instrumentales*, qu'il associe aux différentes façons d'organiser l'apprentissage des étudiants en interaction avec des outils informatiques. Les étudiants regardent-ils le professeur créer et/ou utiliser un outil ? Sont-ils invités à créer et/ou à utiliser un outil en classe devant leurs pairs ? Sont-ils dans un environnement de laboratoire où ils créent et/ou ils utilisent un outil pour compléter des activités interactives ? Trouche (2004) note aussi qu'une orchestration peut viser des niveaux différents de la genèse instrumentale. Le professeur cherche-t-il à enseigner la programmation elle-même (p.e., la syntaxe d'un langage, les fonctions disponibles dans un logiciel) ? Veut-il aider ses étudiants à s'approprier des méthodes de programmation pour accomplir des activités spécifiques ? Les étudiants sont-ils amenés à voir la programmation comme une partie intégrante du travail d'un mathématicien ? Enfin, il est clair que, pour le mathématicien, l'intégration de la programmation est plus complexe dans le contexte d'enseignement, car il doit alors penser aux étudiants.

2.3 La programmation en pratique

Dans son travail, Trouche (2004) parle aussi de la complexité des schèmes d'utilisation associés à un artefact donné. Il explique que ces schèmes sont constitués de plusieurs composantes : les buts, les règles d'action, les conceptions de l'utilisateur et les possibilités associées à l'utilisation. Il propose de plus que ces schèmes sont en même temps construits par un individu et par toute une communauté de pratique auquel l'individu appartient. Cette section sert à introduire un cadre bien développé pour capturer cette complexité. Les composantes des schèmes sont ainsi associées aux *pratiques* et l'aspect social de leur construction aux *contextes institutionnels.* Après avoir précisé les pratiques qui nous intéressent dans cette recherche, nous présentons ces notions qui nous aideront à y situer la programmation.

2.3.1 Les pratiques de qui ?

Si le but de notre projet est de mieux comprendre et de comparer l'intégration de la programmation dans la recherche et dans l'enseignement des mathématiciens, ceci soulève un problème subtil. L'enseignement possède naturellement deux points de vue distincts : celui de l'enseignant et celui de ses étudiants. Lequel devons-nous prendre ?

Après avoir demandé à des mathématiciens de décrire les intersections qu'ils observent entre « chercher » et « enseigner », il nous est devenu clair que la plupart ne voient aucune similitude fondamentale entre ces deux activités. Une de nos participantes explique :

> In one, the focus is on trying to help the students learn. [...] When I'm doing research, I'm developing new concepts. [...] these are very different activities. In one, I know where I want them to go, I know exactly how to take them there. In the other, I kind of know where I'd like to go, though I'm not sure it's going to work out, and I have to try a bunch of different things.

L'élément de recherche et de découverte n'existe pas pour l'enseignant à l'égard des mathématiques qu'il enseigne ; il peut néanmoins exister du point de vue de l'étudiant qui est mis en relation avec ces concepts pour la première fois. Cette même mathématicienne dit que pour les étudiants « it may seem like an open-ended exercise, and that's fine, right? Because I think they learn better if they feel like they're walking into the unknown a little bit. »

En définitive, il nous semble plus approprié de comparer ce que les mathématiciens font en recherche à ce qu'ils font faire à leurs étudiants ; autrement dit, notre mémoire se concentrera sur la comparaison entre « chercher » et « apprendre », en regardant le volet enseignement de la pratique du mathématicien de la perspective de ses étudiants. En effet, si nous faisons l'hypothèse qu'une formation universitaire en mathématiques devrait refléter le travail des mathématiciens professionnels[2], l'examen des similitudes et des différences entre les pratiques en recherche et les pratiques en apprentissage (au lieu des pratiques en enseignement) pourrait être beaucoup plus pertinent et intéressant.

Mais comment comparer les façons de faire de la recherche mathématique à celles d'apprendre des mathématiques ? Pour cela, nous avons choisi de mettre à contribution le concept de *pratique*, en nous référant à la théorie anthropologique de la didactique (la TAD) de Chevallard (1998).

2.3.2 La TAD : un cadre pour décrire les pratiques

Avec cette approche, autant l'activité mathématique d'un professionnel au sein d'un groupe de recherche ou de développement que celle d'un élève ou d'un étudiant dans une institution scolaire peuvent être décrites à travers la notion de *praxéologie* : un modèle où chaque *pratique* inclut un savoir-faire (la *praxis*) et les discours (le *logos*) qui décrivent, légitiment, expliquent et produisent la praxis. La praxis se divise en deux composantes : les *tâches* (les choses à accomplir) et les *techniques* (les méthodes employées pour accomplir les tâches). Par exemple, si la tâche est de résoudre un système d'équations différentielles, on pourrait décider d'utiliser une fonction préprogrammée en *Mathematica* pour le faire, ce qui serait la technique. Similairement, le logos peut se situer sur différents niveaux ; Chevallard (1998) considère comme niveaux distincts du discours les *technologies* (les justifications pour les méthodes choisies) et les *théories* (les fondements de la technologie). L'utilisation d'une technique informatique pour résoudre un système d'équations différentielles peut se justifier

[2] Par « mathématiciens professionnels » nous entendons les gens qui utilisent les mathématiques parmi leurs principaux outils de travail. Cette définition inclut les mathématiciens universitaires, mais également bien d'autres, par exemple des chercheurs dans l'industrie.

par sa vitesse et sa précision en comparaison à une technique à la main. Mais le choix d'une technique plutôt qu'une autre dépend aussi des caractéristiques du système à résoudre et repose alors sur les connaissances qu'on a de certaines théories mathématiques et informatiques : ainsi la solution analytique d'un système d'équations différentielles linéaire à coefficients constants peut être trouvée par la fonction *Mathematica* « DSolve », tandis qu'un système non linéaire nécessitant des solutions numériques peut être résolu avec « NDSolve » ou avec un algorithme qu'on choisira de programmer soi-même. Enfin, ce quadruplet de variables (tâche, technique, technologie, théorie) forme un modèle fondamental pour identifier les éléments qui distinguent les pratiques dans un contexte donné, au sein d'une même institution donnée. Dans notre cas, nous l'utiliserons pour décrire les pratiques des mathématiciens (contexte de recherche) et celles de leurs étudiants (contexte d'apprentissage).

À cause de l'ambigüité du mot « technologie », souvent employé comme synonyme d' « outil informatique », et de l'absence de théories formelles en éducation dans le bagage des mathématiciens, nous avons préféré considérer tout discours expliquant la praxis sous la bannière « justification ». Nous avons décidé néanmoins de distinguer les justifications *pragmatiques* (concernant le potentiel productif d'une technique) et *épistémiques* (concernant le potentiel d'une technique pour contribuer à la compréhension des objets impliqués).

Artigue (2002) suggère que s'il est facile de reconnaître la valeur pragmatique des techniques informatiques, il est peut-être moins naturel d'en cerner la valeur épistémique. Évidemment, l'utilisation des fonctions « DSolve » et « NDSolve » en *Mathematica* pour calculer la solution d'un système d'équations est très efficace, précise et valide pour tout système. Cette technique de résolution a donc une grande valeur pragmatique. Mais qu'en est-il de sa valeur épistémique ? Une tendance commune est de considérer qu'on apprend davantage en résolvant des équations « à la main ». Artigue (2002) propose plutôt que les techniques informatiques peuvent aussi avoir une grande valeur épistémique. Par exemple, l'analyse en classe des fonctions différentes d'un outil informatique (p.e., « DSolve » et « NDSolve ») peut mener à une discussion mathématique riche (p.e., concernant une appréciation des apports respectifs du calcul exact et du calcul approché plutôt que la dévalorisation du second en faveur du premier comme c'est souvent le cas dans la culture mathématique institutionnelle). Les techniques informatiques, qui favorisent le recours aux

paramètres, permettent également de multiplier les observations afin d'accéder à des généralisations ; elles pourraient donc avoir une valeur épistémique autant en recherche qu'en apprentissage.

Enfin, cette distinction des types de justifications nous apparaît importante au regard du développement du répertoire des techniques, tant chez les mathématiciens que chez leurs étudiants. Après tout, selon Artigue (2002), le statut de certaines pratiques dans une communauté peut dépendre de telles justifications et de leur type. Elle note aussi que la nécessité de donner ces justifications peut même disparaître dans les cas où une pratique est devenue « normalisée » au sein d'une institution spécifique.

2.3.3 L'organisation institutionnelle des pratiques

L'idée des pratiques « normalisées » nous mène à un élément crucial de la théorie de Chevallard (1998) : les pratiques des individus n'existent pas toutes seules ; elles vivent dans des institutions sociales qui les encadrent, les développent et les promeuvent. L'intégration de la programmation dans les pratiques des mathématiciens et dans celles de leurs étudiants est donc un phénomène social fortement influencée par les institutions dans lesquelles elle se produit. Une institution importante dans notre cas est la communauté mathématique, qui peut représenter l'ensemble de tous les mathématiciens ou être déclinée selon le groupe de mathématiciens auquel on s'associe (p.e., tous ceux qui travaillent dans un même champ d'études). Ainsi, la société en général, l'université ou le département où travaille un mathématicien, de même que le regroupement de professeurs et de chargés de cours qui enseignent dans un même programme (ou curriculum) sont autant d'institutions à considérer. Chevallard (2002) reconnaît toutes ces institutions dans sa hiérarchie des niveaux qui déterminent des organisations mathématiques et didactiques (Figure 2.1 ci-après ; nous y avons fait des regroupements et des ajouts qui nous seront utiles). Ce travail nous incite à nous rappeler toujours que ces contextes institutionnels ne sont pas isolés : ils interagissent au sein d'un système complexe. Par exemple, une contrainte à un niveau peut influencer le développement des contraintes à un autre niveau.

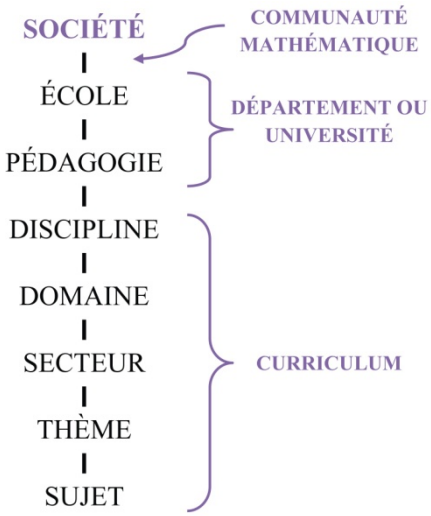


Figure 2.1 Les niveaux de détermination de Chevallard (2002), avec les aspects qui nous intéressent en violet.

Les contraintes et les leviers qui viennent avec les institutions conduisent à la normalisation de certaines pratiques au détriment d'autres et contribuent ainsi à définir « l'activité mathématique », au sens où on l'entend dans une institution donnée (Artigue, 1997). Par exemple, à l'époque où seules les techniques à la main étaient officiellement reconnues en mathématiques et où les techniques informatiques étaient marginalisées, cette organisation donnait lieu à la perception générale selon laquelle les vrais mathématiciens n'utilisent pas l'ordinateur (Bailey et Borwein, 2005). Autrement dit, l'activité mathématique reconnue ne faisait pas appel à des outils informatiques. Par ailleurs, l'adoption croissante de nouvelles pratiques par des mathématiciens peut influencer à son tour les institutions auxquelles ils sont associés. Artigue (2002) explique, par exemple, que si un enseignant décide d'intégrer un outil informatique dans un cours, il doit consacrer du temps à l'introduction des caractéristiques spécifiques de l'outil ; ceci pourrait influencer ses propres pratiques. À long terme, les pratiques innovatrices de cet enseignant, si jugées efficaces, pourraient mener à leur reconnaissance institutionnelle.

Pour décrire le statut variable des pratiques selon l'impact des contextes institutionnels, nous ajoutons à notre cadre théorique des éléments du travail de Morrissette (2011). En analysant les manières de conduire l'évaluation formative au niveau primaire, cette chercheuse a distingué trois types de pratiques dans une profession donnée :

1) les *pratiques partagées*, qui font partie intime de la culture professionnelle et qui ne sont pas remises en question par conséquent ;

2) les *pratiques admises*, qui ne sont pas nécessairement partagées par tous, mais qui sont acceptées dans la culture puisqu'elles se sont révélées particulièrement efficaces par des praticiens innovateurs ;

3) les *pratiques contestées*, qui ne sont pas acceptées par tous et qui se trouvent alors aux frontières de la culture professionnelle.

Nous proposons d'employer cette classification dans notre analyse des pratiques des mathématiciens en recherche et des pratiques des étudiants en mathématiques. L'identification des pratiques conventionnelles, innovantes ou exceptionnelles dans chaque contexte nous permettra non seulement de fournir une meilleure description et comparaison des pratiques, mais aussi d'explorer leur complexité et de cerner les enjeux correspondants. Après tout, si les *pratiques partagées* représentent les conventions actuelles, la mise en évidence de leur coexistence avec des *pratiques admises* et des *pratiques contestées* pourrait suggérer l'opportunité de réexaminer certains aspects des conventions.

2.4 Nos questions spécifiques

En utilisant les théories et les concepts clés que nous avons introduits, nous reformulons nos objectifs généraux et énonçons les trois questions spécifiques suivantes :

1) Comment la programmation se manifeste-t-elle dans les pratiques des mathématiciens en recherche et dans les pratiques en apprentissage qu'ils planifient pour leurs étudiants ?

2) Quelles sont les similarités et les différences entre les pratiques des mathématiciens et les pratiques qu'ils planifient pour leurs étudiants au regard de la place de la programmation ?

3) Quel est le rôle des contextes institutionnels dans les choix des mathématiciens concernant l'intégration de la programmation dans leurs propres pratiques et dans les pratiques qu'ils planifient pour leurs étudiants ?

Notons que la construction de notre cadre théorique semble d'emblée inclure quelques idées originales, avec notamment l'application de la genèse instrumentale et de la notion de praxéologie au travail des mathématiciens et la classification des pratiques en recherche et en apprentissage en employant la terminologie de Morrissette (2011).

3 Méthodologie : une étude qualitative exploratoire

Nos questions de recherche demandent une analyse en profondeur des contextes où pratiquent des mathématiciens, à la fois chercheurs et professeurs. Pour y répondre, nous avons envisagé une étude exploratoire auprès d'un nombre restreint de mathématiciens. Cette section commence avec la description du processus de recrutement de nos participants. Nous présentons ensuite la méthode de cueillette et d'analyse de nos données qualitatives.

3.1 Les participants

Le recrutement des participants potentiels a commencé à l'été 2014, après l'obtention d'un certificat d'éthique. Nous souhaitions rencontrer principalement des mathématiciens qui intègrent la programmation à leur recherche *et* dans leurs cours. À cause de la liberté académique, nous faisions l'hypothèse que les mathématiciens utilisant des programmes informatiques en recherche seraient plus aptes et enclins à en faire utiliser par leurs étudiants. La comparaison de leurs expériences nous permettrait donc de dégager des tendances intéressantes. Cela dit, nous avons reconnu l'intérêt d'inclure quelques participants qui ne font appel à la programmation que dans l'un des deux contextes (la recherche *ou* l'enseignement). La position unique de tels mathématiciens pourrait nous aider à mieux cerner l'impact potentiel des contextes institutionnels.

Les autres variables considérées pendant le processus de sélection nous ont permis d'atteindre une certaine diversité dans notre échantillon. Nous étions particulièrement attentifs aux institutions dans lesquelles les mathématiciens travaillent. Si le fait d'interviewer quelques mathématiciens provenant d'une même institution pourrait révéler des régularités intéressantes, l'inclusion d'expériences au sein d'institutions différentes était essentielle pour éviter des conclusions biaisées. Nous nous sommes ainsi efforcés à constituer un échantillon diversifié de participants, tant au niveau du genre (homme/femme), qu'au niveau de la langue typiquement parlée, du degré d'expérience, du champ d'études et du lieu de travail. Afin de réduire le biais qui pourrait résulter d'un foyer centré à Montréal, nous avons même cherché à inclure des professeurs d'autres régions du Canada. Les endroits spécifiques furent choisis en fonction de leur facilité d'accès pour la chercheuse principale.

Nous avons communiqué par courriel avec 17 participants potentiels pour les inviter à participer à l'étude (une forme générique du courriel d'invitation est donnée en Annexe A). Nous aimons à penser que le grand taux de réponses positives (14 sur 17) reflète l'importance de nos questions de recherche. Les mathématiciens qui ont accepté notre invitation constituent un groupe diversifié : 3 femmes et 11 hommes d'âges variés, qui travaillent dans un grand éventail de domaines dans 10 universités situées dans 3 provinces canadiennes (Colombie Britannique, Ontario et Québec). Adèle, Alice, Alain, Anthony, Albert, Barbara et Ben (prénoms fictifs) sont les mathématiciens qui font l'essentiel de leur recherche en mathématiques appliquées. Leurs projets développent et appliquent des théories mathématiques pour résoudre des problèmes dits « réels », qui sont liés aux sciences, à l'économie ou à l'ingénierie. Nous avons distingué deux mathématiciens de ce groupe par un prénom commençant par B afin de souligner que leur travail est parfois plus « pur ». De façon similaire, nous avons appelé les participants qui travaillent principalement en mathématiques pures : Nathan, Norman, Olivier, Omar, Paul, Pierre et Philippe. Quelques-uns de ces mathématiciens ont travaillé sur des projets appliqués (les N plus que les O et les O plus que les P), mais ces mathématiciens tendent à étudier des concepts plus abstraits dans l'un ou l'autre des champs suivants : probabilités, analyse, algèbre et géométrie.

3.2 La cueillette de données

La cueillette de données a commencé à l'été 2014 et a duré jusqu'à la fin de l'automne 2014. Afin de développer une compréhension profonde des pratiques des 14 participants et de leurs étudiants, nous avons mené des entrevues individuelles.

3.2.1 Le développement d'un outil

Puisque nous voulions aider nos participants à évoquer des descriptions et des réflexions authentiques, nous avons décidé de suivre en grande partie le modèle de l'entretien d'explicitation (Vermersch, 2006). Selon ce modèle, l'intervieweur est simplement un guide qui amène l'interviewé à entrer et rester dans un état de verbalisation descriptive d'actions spécifiques. Dans son livre, Vermersch explique et justifie plusieurs techniques permettant de conduire ce type d'entrevue. Par exemple, il recommande fortement d'éviter le mot

« pourquoi » pour ne pas mettre l'interviewé sur un mode défensif qui le conduirait à donner des excuses insignifiantes ou à entrer dans une posture de professeur où il expliquerait ce qu'il sait au lieu de décrire ce qu'il fait. C'est le déroulement de ses actions, après tout, qui est « la seule source d'inférences fiables pour mettre en évidence les raisonnements effectivement mis en œuvre (différents de ceux adoptés hors de l'engagement dans l'action) » (Vermersch, 2006, p. 18). En menant nos participants à se rappeler des expériences singulières, à les « revivre », le but ultime était d'obtenir des informations aussi fidèles que possible concernant leurs pratiques. Nous espérions ainsi diminuer le biais qui pourrait découler du fait que nous n'aurions pas le temps d'observer la réalisation de telles pratiques.

Un guide a été développé pour fournir une structure générale d'entrevue qui nous aiderait à atteindre nos objectifs de recherche et à suivre les lignes directrices de Vermersch (2006). Nous en avons validé une première version auprès d'un professeur-chercheur en mathématiques. Cette validation nous a permis de juger de la pertinence et de l'efficacité des questions individuelles et de réfléchir à la structure complète du guide. Elle nous a également donné l'occasion de nous exercer dans la gestion générale d'une entrevue et dans la mise en œuvre des techniques de l'entretien d'explicitation, qui nous étaient nouvelles. Au fur et à mesure des entretiens qui ont suivi, nous avons continué à perfectionner nos techniques d'entrevue et à optimiser notre guide. Un résumé écrit après chaque entrevue nous a aidée à identifier de nouvelles directions pour les entrevues suivantes. La forme finale du guide, présentée en Annexe B, donne une bonne idée des grands thèmes abordés avec chaque mathématicien.

3.2.2 L'instrumentation

Le lieu, le moment et la durée des entrevues ont été déterminés avec chaque participant, selon ses disponibilités. La durée typique d'une entrevue se situait entre une et deux heures. Une fois la rencontre fixée, le mathématicien recevait un courriel contenant un formulaire de consentement qui serait revu et recueilli en début d'entrevue (voir Annexe C). Ce courriel servait aussi à lui donner des indications générales concernant le déroulement de l'entrevue.

Si le guide d'entrevue semble indiquer une approche très rigide, nous n'avons pas posé chaque question à chaque mathématicien dans l'ordre indiqué. Sauf pour l'introduction et la conclusion, qui ont été effectuées de la même façon pour chaque entrevue, le document que nous avons construit a vraiment servi de *guide.* En employant les différents types de relance décrits par Vermersch (2006), nous avons mené des entrevues *semi-dirigées*. D'une part, nous sommes toujours restée ouverte à suivre la direction naturelle des idées d'un participant ; une telle approche permettait de réduire l'effet de nos interventions sur ses réponses et d'ouvrir à l'exploration de thèmes inattendus. D'autre part, nous nous sommes aussi assurée d'obtenir le point de vue du mathématicien sur la plupart des sujets suivants :

1) sa propre histoire avec la programmation ;
2) l'utilisation (ou non) des programmes informatiques dans sa recherche ;
3) la programmation (ou non) par ses étudiants aux cycles supérieurs ;
4) l'intégration (ou non) de la programmation dans ses cours ;
5) la place idéale de la programmation dans un premier cycle en mathématiques ;
6) l'impact des contextes institutionnels sur (1) à (5) ;
7) les relations entre la recherche, l'enseignement et l'apprentissage ;
8) les relations entre les mathématiques et la programmation.

Nous avons aussi demandé à chaque participant de partager de vraies ressources (p.e., des programmes informatiques, des activités didactiques) qu'il avait créées dans les contextes de sa recherche et de son enseignement. En plus de nous fournir des détails intéressants sur leurs pratiques, un tel partage de ressources avait l'avantage de pouvoir aider les participants à se souvenir de leurs expériences et, par conséquent, à les décrire.

Pour faciliter le traitement de nos données, les entrevues ont toutes été enregistrées. Nous avons aussi pris quelques notes par écrit pour garder une trace des idées marquantes et des actions non verbales des interviewés. Ces notes se sont révélées particulièrement importantes aux moments où les mathématiciens étaient en train de partager des ressources. Lorsque des ressources étaient montrées pendant une entrevue, nous avons demandé au participant de nous les envoyer par courriel plus tard.

3.3 Le traitement de données

Pour traiter nos données qualitatives, nous avons combiné une analyse par catégories (Van der Maren, 1996) et une analyse thématique (Paillé et Mucchielli, 2010). Après la transcription fidèle de chaque entrevue, nous avons effectué un codage mixte afin d'y identifier les idées principales. Une comparaison du codage fait sur deux entrevues similaires nous a permis d'établir un arbre préliminaire de codes, mais nous avons ajusté l'arbre au fur et à mesure du codage des autres entrevues. Enfin, les codes que nous avons utilisés peuvent être regroupés sous trois grands thèmes : la description des pratiques (en recherche ou en apprentissage, réelles ou souhaitées), l'impact des contextes institutionnels sur les pratiques (en recherche ou en apprentissage) et les réflexions plus générales (p.e., sur les relations entre les mathématiques et la programmation). L'Annexe D présente une liste des codes, leurs définitions et des exemples d'extraits codés.

Nous avons mené l'analyse des éléments codés en variant les approches. Nous avons d'abord procédé à une caractérisation supplémentaire, basée sur le modèle de Chevallard (1998), des unités concernant les pratiques impliquant la programmation. Pour chaque exemple spécifique décrit par les interviewés, les *tâches*, les *techniques* et les *justifications* correspondantes ont été identifiées (voir Annexe D pour quelques exemples). Une telle caractérisation, faite pour nous aider à aborder notre première question de recherche, nous permettait de préciser l'intégration de la programmation dans les pratiques des mathématiciens et de leurs étudiants. Nous avons ensuite comparé les caractérisations des pratiques selon les différents secteurs de recherche (mathématiques appliquées et mathématiques pures) et les différents types de pratiques (pratiques en recherche et pratiques en apprentissage). Ces comparaisons visaient le regroupement des pratiques sous des catégories appropriées afin d'élucider notre deuxième question de recherche. Pour traiter notre troisième question, nous avons étudié soigneusement les extraits du discours des mathématiciens que nous avions associés aux différents codes représentant des variables institutionnelles. Enfin, étant donné la nature exploratoire de notre étude, plusieurs lectures additionnelles des éléments codés (et des entrevues) ont été effectuées pour enrichir nos résultats.

Tout au long du processus d'analyse, nous avons toujours gardé à l'esprit nos tendances naturelles qui pouvaient entraîner certains biais. Par exemple, nous nous sommes rappelée constamment que « la description de tout » est impossible et que les cas qui contredisent nos intuitions initiales sont aussi importants que les autres. Nous nous sommes adaptée aussi à la limite de temps inhérente à la réalisation d'une maîtrise : au lieu d'essayer d'analyser sous tous les angles possibles la grande quantité de données que nous avons collectées, nous avons favorisé une analyse approfondie des aspects les plus pertinents afin d'arriver à un ensemble cohérent de résultats riches. Par conséquent, certains éléments périphériques des données (p.e., l'histoire des participants avec la programmation et la programmation de leurs étudiants aux cycles supérieurs) n'ont été utilisés que dans la mesure où ils permettaient de mieux répondre à nos questions spécifiques de recherche.

3.4 La nature bilingue de l'étude

Un aspect plutôt unique de notre étude est son bilinguisme : le français et l'anglais apparaissent également dans les langues d'entrevue. Réaliser une étude complètement bilingue était un objectif ambitieux et courageux, compte tenu de notre appropriation concomitante du français. Il y a eu des moments où nous avions du mal à suivre les interviewés qui s'exprimaient en français. Pour garantir des transcriptions de qualité, nous avons souvent eu à écouter les enregistrements plusieurs fois et s'il restait encore des doutes, nous avons obtenu de l'aide. Avec le soutien constant de nos directeurs francophones, nous avons pu assurer la fidélité de nos données et faire de ce caractère bilingue un levier à la fois pour l'étude (avec la diversité de l'échantillon considéré) et pour notre propre formation (avec l'occasion de développer autant notre propre réseau pancanadien de contacts que nos compétences linguistiques). Nous espérons enfin que le caractère bilingue de notre recherche en facilitera la diffusion auprès d'une plus grande communauté de chercheurs.

4 Description des pratiques

Ce chapitre débute l'analyse des entrevues réalisées dans notre étude. La section 4.2.1 aborde notre première question de recherche. Nous y décrivons la place de la programmation dans les pratiques en recherche des 14 mathématiciens que nous avons interviewés. Nous présentons ensuite, à la section 4.3, les pratiques en apprentissage impliquant la programmation qu'ils proposent pour leurs étudiants. Mais, avant d'expliciter telles pratiques, la section 4.1 revient sur le sens de « la programmation », en présentant les significations que nos participants accordent à ce mot.

4.1 La programmation : clarification d'une définition

Au début de ce projet (voir la section 2.1), nous avons essayé de cerner sommairement les activités qui pouvaient être vues comme de « la programmation » et qui devraient par conséquent être considérées dans ce mémoire. Clairement, la résolution numérique d'un système d'équations différentielles nécessitant un nouvel algorithme, du codage et la comparaison des résultats informatiques avec certaines propriétés connues implique la programmation. Mais qu'en est-il de l'obtention, à l'aide de *Mathematica*, du calcul d'une intégrale particulièrement récalcitrante ? Ou du traçage d'une fonction compliquée pour en connaître le comportement autour d'un point donné ?

Afin qu'ils nous aident à mieux circonscrire l'activité de programmation, nous avons décidé de laisser aux participants l'interprétation du mot « programmation » pendant les entrevues. Nous avons même demandé explicitement à certains mathématiciens de partager leur interprétation. En analysant l'ensemble de perceptions qui a émergé, nous identifions un groupe de tâches qui est reconnu par pratiquement tous les mathématiciens comme faisant partie de l'activité de programmation. Nous commençons par la description de ces tâches pour développer une définition générale de la programmation. Suit une discussion des activités qui ne sont pas considérées unanimement comme des activités de programmation et qui exposent donc des subtilités de la définition premièrement construite. La section se termine avec une présentation des réponses à une question qui pousse plus loin la caractérisation de la programmation : « quel est le lien entre programmer et faire des mathématiques ? »

4.1.1 Une description de l'activité de programmation

La programmation constitue rarement l'activité principale d'un projet de recherche en mathématiques. C'est plutôt la tâche que cherche à accomplir le chercheur qui peut l'amener à vouloir utiliser l'outil informatique. Le recours à cet outil peut être simplement utile dans la mesure où il permet une économie de temps et d'effort, mais il peut aussi être absolument nécessaire en raison de la difficulté de la tâche à accomplir. L'activité de programmation a donc pour but de créer ce programme, qui est ensuite utilisé pour accomplir cette tâche.

En combinant toutes les idées discutées par nos participants, la programmation en mathématiques émerge comme une activité qui implique trois tâches : le développement d'un algorithme, le codage de l'algorithme, et la vérification et la validation du programme. Si ces tâches ne sont pas faites nécessairement dans cet ordre et si elles peuvent se succéder dans un cycle itératif, la première étape est habituellement l'élaboration d'un algorithme.

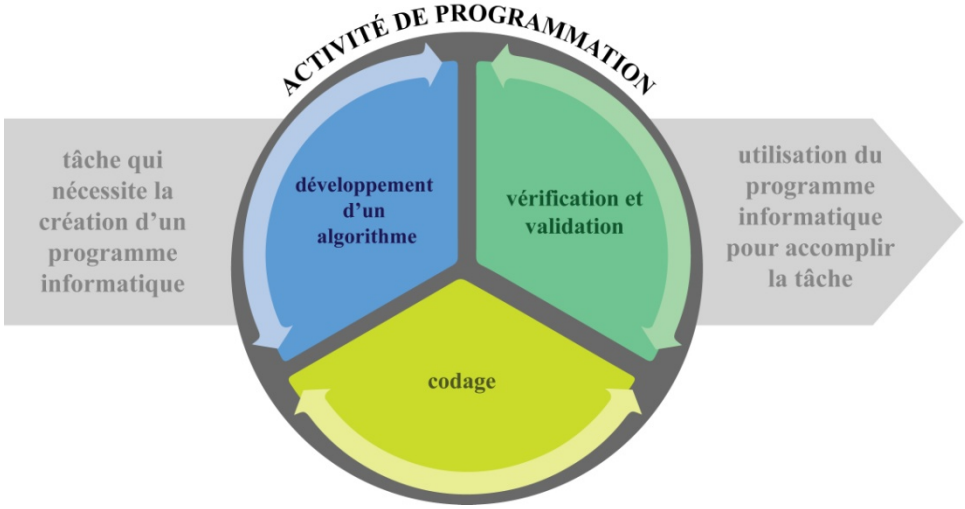


Figure 4.1 Une visualisation de l'activité de programmation.

Le développement d'un algorithme est typiquement mené sans l'ordinateur. Il comprend l'identification d'une série d'actions qui accomplira la tâche. Il comprend également le choix des structures de données à adopter (listes, matrices, arbres, réseaux, pointeurs, etc.), le type de programmation (procédurale, orientée objets, par manipulations de listes, etc.), le format des données initiales et des résultats, etc. Implicitement, le choix du langage est aussi fait à cette étape et, en raison des structures et des approches qu'il permet, ce choix contribuera à influencer l'algorithme.

Le codage est l'étape de traduction de l'algorithme dans le langage choisi. Il est exécuté à l'ordinateur. Notons l'ensemble varié des langages utilisés par nos participants : *Mathematica, MATLAB, Visual Basic.Net, Sage, R, C, C++, C#, Fortran* et *Python.*

Enfin *la vérification et la validation* comprennent la compilation du programme et les corrections des erreurs de syntaxe, les comparaisons des résultats obtenus par l'ordinateur avec des exemples élémentaires connus, bref l'ensemble des tests pour déterminer si le programme fait ce qu'il doit faire d'une façon adéquate ou non.

Le processus de cette activité de programmation n'est pas du tout linéaire. Selon la complexité du programme, le codage, la vérification et la validation peuvent commencer avant que la totalité de l'algorithme n'ait été fixée. Par exemple, pour écrire des programmes simples, des mathématiciens peuvent mener de front le développement d'un algorithme et son codage. Dans le cas des programmes complexes, la construction se fera typiquement par le développement et la combinaison de plusieurs petits programmes. De plus, pendant l'étape de vérification et de validation, le chercheur pourra remarquer qu'il y a une erreur dans le code ou dans l'algorithme, ce qui le forcera à revoir ces éléments. Plusieurs mathématiciens parlent aussi d'analyser certaines propriétés de leur algorithme (p.e., son efficacité) avant de le coder. Ainsi, il est possible qu'un programmeur passe par une étape de validation de l'algorithme avant même de commencer le codage. Enfin, il n'est pas rare qu'un chercheur construise, teste et améliore plusieurs algorithmes avant d'arriver à son programme idéal.

La construction d'un programme, son utilisation et les autres tâches liées à un projet de recherche sont menées elles-mêmes dans un ordre non linéaire. Le chercheur pourra alterner entre des moments de réflexion loin de l'ordinateur et de nouvelles phases d'utilisation et de modification de son programme. Un seul programme pourra même être adapté aux buts d'autres projets de recherche, y simplifiant l'activité de programmation requise.

La section 4.2 détaille comment nos participants s'attaquent à chacune des sous-tâches de l'activité de programmation selon leur domaine et leurs buts. Elle discute aussi d'autres tâches plus près du génie logiciel (p.e., l'entretien des programmes au fil des ans) qui n'occupent une place essentielle que pour certains mathématiciens. Pour l'instant, l'ensemble des trois phases décrites ci-avant formera le cœur de notre définition de la programmation.

4.1.2 Les frontières de l'activité de programmation

Parfois un chercheur peut réaliser si rapidement l'une ou plusieurs des trois tâches associées à la programmation qu'il ne qualifiera pas d' « activité de programmation » son utilisation de l'ordinateur. Pour un autre chercheur, par exemple un programmeur moins chevronné, le même exercice pourrait lui demander plus d'effort et faire en sorte que ce chercheur n'hésite pas à le décrire comme de la programmation. Il n'est donc pas surprenant que ce qui constitue vraiment l' « activité de programmation » ne fasse pas l'unanimité.

Les logiciels de calcul symbolique, à cause de leurs vastes bibliothèques de fonctions préprogrammées, permettent en quelques lignes de code de réaliser d'énormes calculs. Il est alors difficile de trancher à partir de quand leur usage constitue de la programmation. Par exemple, pour plusieurs mathématiciens, l'utilisation de ces logiciels (ou d'une calculatrice graphique) pour faire tracer le graphe d'une fonction ne constitue pas vraiment de la programmation. De la même façon, l'emploi de codes écrits par quelqu'un d'autre est rarement reconnue comme une activité de programmation. Un des mathématiciens apporte tout de même une nuance intéressante à travers le questionnement suivant :

> So, running packages in *Sage* or whatever, is that programming?
> *What do you think ?*
> Sometimes it is to me because I have to figure out what the code is.

Un autre chercheur évoque une activité de programmation en *Mathematica* qui lui a permis une avancée majeure dans sa recherche. Comme nous avons pu reproduire son résultat en écrivant quelques lignes de code seulement (voir la section 4.2.2) plusieurs des mathématiciens rencontrés se refuseraient à le considérer comme de la vraie programmation ; à l'inverse, d'autres participants ne verraient aucun problème à inclure pareille utilisation simple de *Mathematica* parmi les activités de programmation. Un des participants affirme : « Pour moi, on peut programmer un outil de manipulation symbolique. Quand je fais des dessins, j'ai l'impression de programmer. Quand j'énumère des choses, j'ai l'impression de programmer. »

Certains mathématiciens rencontrés adoptent une définition encore plus large de la programmation. Ils emploient souvent la phrase « utiliser l'ordinateur » comme synonyme de

« programmer ». Selon cet usage, un nombre énorme d'activités seront classifiées comme « programmation » : l'écriture de documents en *LaTeX*, la mise sur pied d'un site internet pour présenter des résultats mathématiques, la création des documents de présentation pour des conférences, etc. Un des mathématiciens propose même que les constructions géométriques dans un logiciel tel que *Geometer's Sketchpad (GSP)* puissent être considérées comme une forme de programmation.

Pour mieux comprendre le sens de cette dernière proposition, considérons une tâche que ce mathématicien donne à ses étudiants. Dans son cours d'introduction aux géométries, les étudiants doivent construire un losange en *GSP*. Les images ci-dessous montrent une méthode possible pour arriver à cette construction. Même si les étudiants n'écrivent aucune ligne de

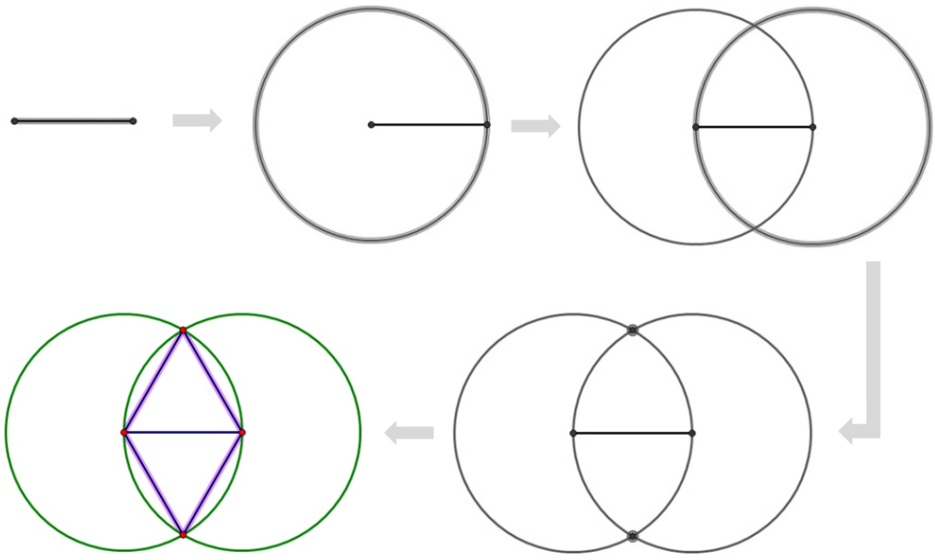


Figure 4.2 La construction d'un losange en *GSP*.

code, les tâches qu'ils accomplissent partagent des caractéristiques avec l'activité de programmation telle que nous l'avons décrite. Premièrement, ils déterminent un ensemble ordonné d'étapes qui emploie les capacités de *GSP* pour créer un losange. Le mathématicien suggère que cet algorithme n'est pas nécessairement intuitif pour les étudiants ; ils passent souvent du temps en dehors de l'ordinateur pour y arriver. Une fois déterminées, les étapes sont effectuées en *GSP*, l'une après l'autre, dans l'ordre nécessaire, avec les fonctions disponibles et selon les règles strictes du logiciel ; même si elle s'appuie sur des menus iconographiques, cette tâche n'est pas bien différente du codage d'un algorithme dans un

langage de programmation. Ensuite, *GSP* permet aux étudiants de vérifier très facilement s'ils ont créé un losange : en faisant afficher, par exemple, les longueurs des côtés et la mesure des angles intérieurs du polygone qu'ils ont produit. Pour valider la stabilité de leur construction, les étudiants peuvent faire glisser les points et les segments de différentes façons, s'assurer que les quatre côtés restent congrus et examiner ce qui se passe avec les angles. Ce jeu sur les paramètres de l'objet est important pour juger de sa généralité et donc de l'utilité du « programme » développé. Dans notre cas, il nous montre que la méthode que nous avons suivie produit un losange très spécifique, qui est constitué de deux triangles équilatéraux (observer les angles à la Figure 4.3). Évidemment, si le but était de « programmer » un objet qui puisse représenter un losange quelconque, nous aurions à développer un nouvel algorithme.

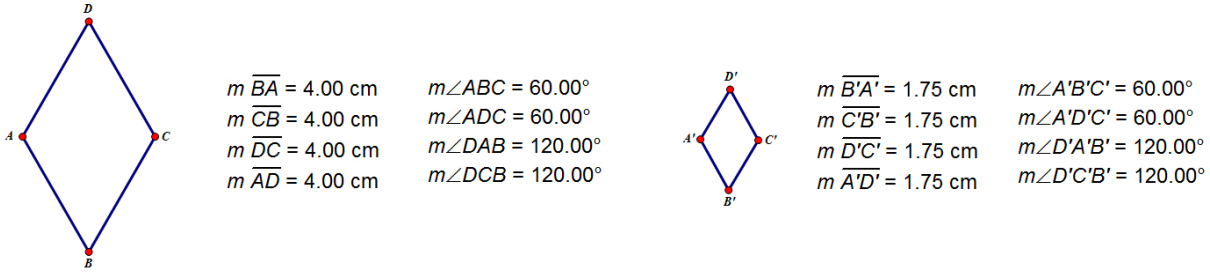


Figure 4.3 La vérification et la validation de la construction d'un losange en *GSP*.

Finalement, les mathématiciens de notre étude ne donnent pas toujours le même poids à chacune des tâches que nous avons identifiées (développement de l'algorithme, codage, vérification et validation). L'un d'entre eux accorde une plus grande attention à la première étape : « Apprendre à programmer, c'est apprendre à transformer un énoncé mathématique en algorithme. » Un autre demande : « Programming means implementing an algorithm, right? » Quelques mathématiciens semblent en effet réduire la définition de la programmation à l'étape de codage, excluant ainsi l'élaboration et la validation de l'algorithme, alors que la plupart des participants incluent toutes les étapes.

Ces diverses opinions donnent donc à l'activité de programmation des « frontières » floues. Étant donné la diversité et l'importance des tâches mathématiques pouvant être réalisées par l'ordinateur, ce flou était prévisible. Il faudra cependant garder en tête cette diversité dans notre analyse.

4.1.3 Une activité mathématique ?

La circonscription de l'activité de programmation par un mathématicien paraît intimement liée à la perception qu'il a de cette activité parmi les autres tâches liées à la recherche. Comment la programmation se compare-t-elle alors à ces autres tâches ? Il est évident que tous les mathématiciens voient la programmation comme une façon de créer des outils pour les aider dans l'accomplissement de leurs activités plus larges de recherches ; mais en concluent-ils pour autant que l'activité de programmation est elle-même une activité mathématique ? Comme nous le souffle une de nos participantes, il nous faut reconnaître que « the difficulty with answering this immediately is the fact that doing mathematics is a very poorly defined activity. » Évidemment, la réponse d'un mathématicien dépend de sa propre définition des mathématiques. Même si nous n'avons pas pu aborder cette question avec tous nos participants, les réponses obtenues permettent d'identifier certaines positions communes.

En général, l'élaboration d'un algorithme est perçue comme une activité semblable à la résolution d'un problème ou à la construction d'une preuve, deux activités indéniablement mathématiques. C'est donc, pour la majorité des mathématiciens rencontrés, une activité très mathématique. Mais les réponses de quelques participants demandent de considérer le type de raisonnement impliqué dans la tâche avant de sauter à son classement comme une activité mathématique. Certains participants insistent sur le fait que la recherche du meilleur algorithme doit exiger un raisonnement complexe, profond ou original pour être assimilable à « faire des mathématiques ». Par contre, une mathématicienne suggère que la nécessité d'un autre niveau de raisonnement distingue l'activité de programmation de l'activité de « faire des mathématiques » : l'informaticien doit toujours penser en termes physiques, matériels, liés à la machine, et pas seulement en termes abstraits et théoriques. Un exemple concret de cette idée est donné par un autre participant. Dans l'un de ses projets, il développe un algorithme qui est parfaitement valide en théorie, mais qui se révèle complètement inutile une fois codé à cause de son instabilité. Il explique : « Bien que notre algorithme, comme preuve mathématique, était excellent, ça ne fonctionnait pas pour une vraie programmation. Pourquoi ? Parce qu'il était extrêmement sensible aux données initiales. [...] c'est vraiment deux choses différentes : d'appliquer quelque chose en réalité et de démontrer. »

Les mathématiciens sont généralement plus enclins à refuser au codage le statut d'activité mathématique. Un mathématicien est catégorique : « Si on prend un algorithme déjà existant et on fait juste l'implémenter, ce n'est pas une activité mathématique. » Cependant, après un peu de réflexion, un autre mathématicien souligne le caractère astreignant et formateur de cette tâche : « [Le codage], c'est aussi quelque chose qui est rigoureux. [...] Ça suit des règles logiques très claires, [...] c'est même plus clair qu'en mathématiques. Et aussi une seule erreur peut détruire tout. » Un autre participant va plus loin en mentionnant que, quand on traduit les mathématiques en langage informatique, on est en train de faire un processus d'abstraction similaire aux processus qu'on mobilise en faisant ou en apprenant des mathématiques.

Nous ne trouvons pas non plus de consensus concernant la nature mathématique de l'étape de vérification et de validation. Cette tâche se déroule à plusieurs niveaux. La correction d'erreurs de syntaxe est très mécanique et peu y reconnaissent une activité mathématique. On peut se demander toutefois si les mathématiciens considéreraient la vérification d'un long calcul algébrique comme une activité mathématique : ce travail demande rigueur et attention, mais est aussi purement mécanique. Quand la vérification et la validation d'un programme vont au-delà des corrections syntaxiques, ce sont peut-être des processus plus mathématiques. Un mathématicien explique : « Writing a computer program is [...] like writing the logical steps of a proof. And then to check whether or not it works, you see if your program runs, just like whether or not your proof is right. » Autrement dit, l'identification et la correction des erreurs plus logiques concernant l'algorithme, qui pourront s'apparenter à la correction d'une preuve, sont des activités mathématiques. D'autres mathématiciens de l'étude insistent sur l'importance des connaissances mathématiques pour la validation des résultats d'un programme dont l'algorithme et le codage ont été déjà vérifiés. Par exemple, l'application de certaines techniques d'analyse à un système d'équations pour vérifier le comportement observé dans des simulations informatiques est vue comme une activité mathématique. Valider des propriétés d'un algorithme ou des résultats observés par l'écriture des preuves formelles est également considéré comme faire des mathématiques. En effet, comme nous le verrons plus tard (voir la section 4.2), ces activités de vérification et de validation font partie essentielle du travail de plusieurs participants.

Une mathématicienne nous invite à faire un parallèle entre l'activité de programmation dans son ensemble et l'activité mathématique. Elle suggère que l'activité mathématique consiste principalement à manipuler des affirmations logiques ou des suites d'affirmations. L'utilisation d'un ensemble d'axiomes, de définitions, de résultats en découlant et d'arguments soumis à la logique permet aux mathématiciens de déterminer si un nouvel énoncé est vrai ou non. La programmation, elle, a pour but de faire réaliser une tâche à un ordinateur. Ici, le programmeur brisera le processus en une suite de sous-tâches, chacune d'entre elles devant être vérifiée et validée. L'interconnexion de ces sous-tâches devra également être vérifiée. Ainsi la programmation, comme l'activité mathématique, est très structurée. Et cette idée est reprise par les autres mathématiciens. Un participant résume : « Computer programming requires logical reasoning, assessing correctness, so it requires mathematical ability. »

Mais cette mathématicienne qui nous signale le parallèle entre programmer et « faire des mathématiques » souligne aussi une différence subtile entre ces deux activités : la première activité exige la construction d'un chemin vers une solution, qui est alors suivi par l'ordinateur, tandis qu'en mathématiques, on n'a pas nécessairement besoin de construire la solution pour savoir qu'elle existe. Faire des mathématiques peut parfois être beaucoup plus qualitatif que constructif. Finalement, la mathématicienne en déduit que « computing is a mathematical activity. But, not all mathematical activity is computing. »

Ce court bilan sur la question « est-ce que la programmation est une activité mathématique ? » montre une grande diversité d'idées ; mais enfin, la plupart des participants de notre étude reconnaissent un caractère mathématique dans certains aspects de l'activité de programmation. Ceci étant dit, on n'accorde généralement à la programmation qu'un statut d'outil ou de méthode ; plus qu'une *façon de faire* des mathématiques, elle est vue principalement comme un *moyen pour faire* des mathématiques. Comme nous le verrons plus tard, la position subalterne réservée à la programmation face aux « vraies mathématiques » (les idées, abstraites ou appliquées, et leur validation) peut nous aider à mieux comprendre les différences entre les pratiques des mathématiciens qui impliquent la programmation et celles qu'ils planifient pour leurs étudiants. Nous nous attaquons en premier à la description de telles pratiques.

4.2 Les pratiques en recherche impliquant la programmation

Les mathématiciens de notre étude font appel à l'activité de programmation dans les différentes formes que nous venons de décrire à plusieurs moments distincts dans leur recherche. La plupart de ces moments font partie de la résolution de problèmes spécifiques, soit en mathématiques appliquées, soit en mathématiques pures. Mais quelques-uns de nos participants discutent aussi des moments où ils développent des outils qui seront utiles à la résolution d'une plus grande classe de problèmes. Dans les sections suivantes, nous élaborons les pratiques où les mathématiciens font de la programmation dans la résolution de problèmes dits « appliqués », dans la résolution de problèmes dits « purs », et finalement, dans le développement d'outils.

En analysant nos données, nous avons ressenti le besoin de situer les moments de programmation dans l'ensemble du processus de recherche. Pour ce faire, nous avons d'abord cherché à définir les tâches génériques qui puissent représenter l'ensemble du processus de recherche des participants pour ensuite y analyser la place de la programmation. De plus, nous avons décidé de diviser les problèmes mathématiques en employant la dichotomie commune distinguant entre « mathématiques appliquées » et « mathématiques pures » pour faciliter notre analyse. Si cette distinction traditionnelle ne rend pas toujours justice à la complexité des projets des mathématiciens, lesquels peuvent avoir des aspects appliqués et purs en même temps, elle émergeait néanmoins de façon récurrente dans les discussions que nous avons eues avec les mathématiciens.[3]

[3] Poursuivant dans cette logique, nous nous permettons même de référer aux « mathématiciens appliqués » et aux « mathématiciens purs ».

4.2.1 La résolution de problèmes en mathématiques appliquées

L'activité de programmation s'inscrit naturellement dans la résolution de problèmes en mathématiques appliquées, laquelle s'intéresse principalement à la modélisation mathématique de phénomènes réels. Tous les mathématiciens appliqués rencontrés déclarent avoir besoin de programmer à un moment ou un autre dans leurs projets. Pour identifier ces moments, nous commençons avec l'exploration d'un exemple spécifique à partir duquel nous dégageons des tâches génériques. Nous donnons ensuite une description plus détaillée des pratiques génériques identifiées.

Un exemple illustratif

À un moment dans sa carrière, Alice est approchée par un collègue d'un autre département. Ce scientifique s'intéresse à plusieurs questions concernant les muscles : pourquoi ils se gonflent, comment leur organisation affecte les forces qu'ils peuvent déployer, comment ils se contractent, etc. Il fait appel à Alice pour l'aider à créer un modèle mathématique qui lui permette d'explorer de telles questions. Alice accepte avec intérêt et enthousiasme.

Connaissant très peu sur les muscles, Alice commence en mode d'apprentissage. Elle explique :

> You have to learn about the specifics of the application. Then, as part of that learning, you see whether people have developed mathematical models. You try to assess their strengths and their weaknesses, and then you try to see if the question that your collaborator might have can be answered with the existing models.

Si un modèle adéquat n'existe pas, on doit construire un nouveau modèle en prenant en considération les modèles existants, les éléments du phénomène auxquels on s'intéresse et les observations des données expérimentales. Alice suggère que ce processus de développement d'un modèle peut prendre plusieurs années. Si nous ne pouvons décrire dans le détail de ses équations le modèle complexe qu'elle a construit, nous pouvons tout au moins expliquer le modèle conceptuel sur lequel il s'appuie : le muscle est ainsi associé à un bloc de matière couvert d'un tissu conjonctif et puis tiré. On s'intéresse à des aspects spécifiques dans cette configuration, tels les tensions, les gonflements et les fibres.

Pareil modèle conceptuel est alors développé mathématiquement pour être étudié. C'est à ce moment que l'utilisation de l'ordinateur devient inévitable. Alice explique : « It's highly unlikely that a mathematical model will give you the quadratic formula in the end. It would be nice, but that doesn't happen. And so, computer programming is essential. » Certains chercheurs utilisent des logiciels prédéveloppés pour simuler leurs modèles ; Alice préfère créer ses propres outils, ce qui a des avantages sur l'autre approche, mais est plus exigeant :

> We really have deep rooted control over everything. But it also makes it very difficult to code up. Because it's not just push a button and here it goes. We really have to think about what choices of finite elements we were making, how to design meshes, how to put the equations in, how to import data, and so forth.

Pour faciliter leur travail, Alice et ses collaborateurs utilisent comme point de départ le code d'autres chercheurs qui étudient des matériaux élastiques. Ils passent toutefois beaucoup de temps à modifier l'algorithme, changer le code et vérifier et valider le programme à nouveau. Alice mentionne plusieurs stratégies qu'elle emploie au moment des étapes de vérification et de validation : elle écrit son code par étapes et teste chacune des sous-routines ; elle élimine les termes non linéaires dans le code et travaille sur le problème linéaire plus simple et mieux connu ; elle utilise des techniques d'analyse pour prédire le comportement pour des cas extrêmes, par exemple, des muscles qui n'auraient aucune fibre. À la fin du processus, les chercheurs obtiennent un simulateur en trois dimensions d'un muscle, ses tendons et son tissu conjonctif qu'Alice décrit comme étant « the most complex simulator of its kind. » Le code final est si imposant que lorsque nous lui avons demandé de le voir, Alice était très hésitante : « You want to go through… I mean I can show you, […] I don't know how useful it will be. » Le niveau le plus haut de son code est fait d'environ 5000 lignes, auxquelles se greffent de nombreuses procédures et fonctions.

Les chercheurs peuvent calculer plusieurs variables avec leur simulateur : par exemple, les tensions sur un muscle, l'activation de ses fibres et son gonflement. Ils essaient en premier lieu de répondre aux questions initialement posées par le scientifique. Mais éventuellement, ils se permettent de jouer plus librement avec les paramètres du modèle, ce qui les conduit à une découverte inattendue : dans leurs simulations, les fibres dans les muscles commencent à avoir une structure qui, selon leurs connaissances, n'avait jamais été rapportée. Nous pouvions entendre l'enthousiasme d'Alice lorsqu'elle nous a raconté cet épisode :

> We didn't seek out these [structures], […] we just plotted the stuff. And then we showed it to […] our collaborator, and he said "You know what? I have my other student run this ultrasound experiment with somebody on a bicycle, and we've seen these [structures], but in a human!" [...] That was really awesome.

La validation de ce qu'ils ont observé par de vraies expériences donne assez de confiance aux chercheurs pour communiquer leurs résultats à la communauté mathématique.

Au moment de la communication, les chercheurs décrivent leur modèle et les résultats obtenus. Ils montrent aussi plusieurs graphiques qui illustrent leurs résultats principaux. Ils doivent également discuter de la performance du simulateur utilisé pour arriver à ces résultats. Mais partagent-ils leur programme informatique ? Notons qu'Alice et ses collaborateurs ne considèrent pas avoir terminé leur travail avec leur simulateur. En parlant de ce programme, elle dit : « I think of this as infrastructure. So I'm developing infrastructure, which once it's completely validated and up and running, we should be able to use it, or my colleague should be able to use it over and over again for different muscles. » Durant l'entrevue, Alice mentionne plusieurs autres aspects que son équipe aimait explorer : par exemple, l'évolution des muscles humains à travers le temps et les muscles d'autres espèces pour lesquelles il y a beaucoup de données expérimentales. La mathématicienne insiste néanmoins que finalement, lorsqu'elle aura fait un « nettoyage » du code et l'aura assorti d'une meilleure documentation, elle l'ajoutera à une bibliothèque ouverte où il sera disponible pour d'autres chercheurs. Donc, en plus du modèle mathématique d'un muscle, le programme informatique qui le simule sera ainsi une contribution importante à la communauté.

Les tâches génériques du processus de recherche en mathématiques appliquées

En réfléchissant sur l'évolution de son propre processus de recherche, Alice suggère que « the process of trying to understand a phenomenon, writing a mathematical description, analyzing that description [...] it's so ancient and so reasonable. Why would you tinker with that? » Des chercheurs en didactique des mathématiques (p.e., Blum et coll., 2002 ; Caron et Squalli, 2009) ont tenté de capturer ce processus intemporel de recherche en mathématiques appliquées. Leurs descriptions nous ont offert un point de départ pour distinguer les étapes différentes dans l'exemple d'Alice. En les combinant avec celles identifiées dans les projets décrits par d'autres participants, nous arrivons à une liste de tâches génériques :

1) choix d'un phénomène réel ;
2) développement d'un modèle ;
    a) étude du phénomène et des modèles existants ;
    b) identification des aspects importants ;
    c) analyse de données réelles ;
    d) création du modèle mathématique ;
    e) validation analytique du modèle ;
    f) calcul des paramètres ;
3) programmation du modèle ;
    a) recherche sur les outils informatiques existants ;
    b) développement d'un algorithme ;
    c) codage de l'algorithme ;
    d) vérification et validation du programme ;
4) recherche des solutions et expérimentation avec le modèle ;
5) interprétation et validation des résultats ;
6) communication des résultats et préparation pour le futur.

Dans un projet donné, ces tâches ne procèdent pas nécessairement de façon linéaire, elles ne sont pas toujours distinctes et certaines sont parfois omises. Néanmoins, la liste ci-dessus représente bien l'ensemble de tâches dans les pratiques appliquées décrites par les mathématiciens que nous avons rencontrés. Par conséquent, elle nous fournit une bonne structure pour situer et apprécier la place de la programmation dans leur recherche.

La description des tâches et de la place de la programmation

Dans cette section, nous présentons une description plus générale des tâches génériques que nous venons d'identifier et y situons l'activité de programmation. Nous enrichissons notre discussion avec des éléments des projets d'autres mathématiciens appliqués de notre étude.

Choix d'un phénomène réel (1)

Chaque projet d'application part d'un phénomène réel qui paraît pouvoir bénéficier d'une étude mathématique. Ce choix est souvent fait par une autre personne, soit un mathématicien, un scientifique ou un client, qui a besoin de l'expertise du chercheur. Albert est

contacté par des mathématiciens nécessitant de l'aide pour l'étude de la mécanique céleste ; Alice est approchée par un scientifique pour développer un modèle d'un muscle ; Adèle est embauché par des institutions financières pour faire l'analyse du risque sur le marché ; Alain est recruté par une entreprise pour concevoir une pompe pour le cœur. Mais parfois le mathématicien choisit le problème à étudier selon ses propres intérêts. Les projets de Ben concernant le marché boursier et les projets d'Alain concernant la mécanique des fluides sont des projets initiés par ces chercheurs.

Développement d'un modèle (2)

Pour étudier le phénomène choisi, les mathématiciens en élaborent une description (ou modèle) mathématique. Si comme dans l'exemple d'Alice les chercheurs peuvent choisir de construire un nouveau modèle, ce n'est pas toujours le cas. D'habitude, les mathématiciens font en premier une *étude du phénomène et des modèles existants (a)*. Albert est un chercheur qui emploie des techniques avancées pour mieux comprendre des modèles existants. Dans un projet, il étudie le système suivant, qui vient d'un problème classique en mécanique céleste[4] :

$$x'' = 2y' + x - (1-M)(x+M)r_1^{-3} - M(x-1+M)r_2^{-3}\,;$$

$$(*) \qquad y'' = -2x' + y - (1-M)yr_1^{-3} - Myr_2^{-3}\,;$$

$$z'' = -(1-M)zr_1^{-3} - Mzr_2^{-3}\,;$$

où

$$M = m_2/(m_1+m_2)\,;$$

$$r_1 = \sqrt{(x+M)^2 + y^2 + z^2}\,;$$

$$r_2 = \sqrt{(x-1+M)^2 + y^2 + z^2}\,.$$

[4] Les premières variantes de ce problème remontent aux années 1600, dans les travaux de Kepler et de Newton. Bien avant l'arrivée de l'ordinateur, ce problème était un sujet de recherche pour plusieurs physiciens et mathématiciens ; ils en ont étudié des propriétés et en ont calculé à la main des solutions approchées. Dans les années 1890, avec le travail de Poincaré, la communauté mathématique a commencé à douter de l'existence d'une forme analytique close pour les solutions de (*). L'outil informatique est depuis devenu essentiel à la découverte de la complexité de l'ensemble des solutions à ce problème ancien.

Sans plus d'explication, un modèle décrit comme celui ci-avant peut être difficile à comprendre ; nous avons besoin des définitions de tous les variables et paramètres impliqués, c'est-à-dire les *aspects importants du phénomène identifiés (b)* par les scientifiques qui ont créé le modèle en premier lieu. Dans ce système, on s'intéresse à décrire le mouvement d'un corps (p.e., un vaisseau spatial) qui a une masse négligeable et qui est sous l'influence gravitationnelle de deux corps massifs (P_1 et P_2) se déplaçant sur des orbites circulaires autour de leur centre de masse commun (O). Évidemment, une chose importante est la position

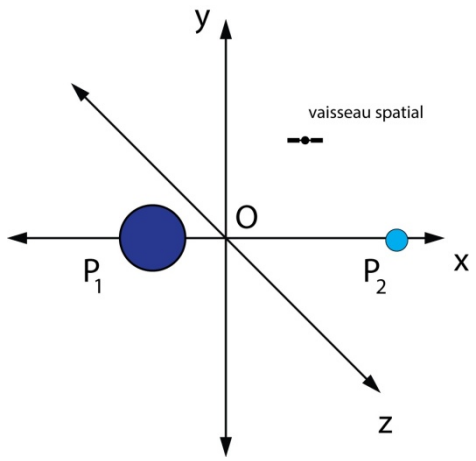


Figure 4.4 La situation modélisée par (*).

(x, y, z) du corps et sa vitesse (x', y', z'). Un autre aspect jugé important est la masse de chacun des deux corps massifs, qu'on note par m_1 et m_2, avec $m_2 \leq m_1$. La détermination des relations entre ces variables et paramètres peut ensuite être traduite par le système d'équations donné en (*), pour *créer un modèle mathématique (d)* qui est très bien connu en mécanique.

Pour déterminer les régularités sur lesquelles la création d'un modèle mathématique est basée, les mathématiciens doivent souvent *analyser des données réelles (c)* du phénomène à l'étude. Ben, par exemple, a étudié longtemps le comportement des titres boursiers avant de découvrir les relations lui permettant aujourd'hui de prédire de grandes pertes dans le marché boursier. Alain, lui, observe plutôt le véritable comportement des fluides dans un laboratoire expérimental qu'il a créé. Même si les sujets des deux mathématiciens sont différents, leur travail à ce point de leur recherche partage une caractéristique cruciale : la nécessité d'utiliser des outils informatiques pour leurs capacités de calcul et de visualisation (deux apports de l'ordinateur mentionnés par Cornu, 1992). L'ordinateur permet aux mathématiciens d'effectuer

rapidement et précisément des calculs numériques sur un grand nombre de données et de représenter efficacement les résultats sous diverses formes ; la programmation peut donc être nécessaire pour des raisons *pragmatiques*. Mais c'est la valeur *épistémique* de la programmation qui est vraiment importante ici : la possibilité de visualiser les résultats à l'aide de tableaux, de graphiques et d'animations peut amener le chercheur à remarquer des régularités intéressantes.

À titre d'exemple, considérons un graphique utilisé par Ben pour expliquer la signification d'un modèle qu'il a développé. Après avoir remarqué qu'il existe une forte corrélation entre le comportement de différents titres sur le marché boursier, Ben s'est demandé : « How many truly independent stocks are there? » Ainsi, il a conçu une méthode qui, en manipulant un très grand nombre de données, estime pour un jour donné le nombre de titres indépendants (le #TI) parmi les 3000 titres sur le marché boursier à New York.

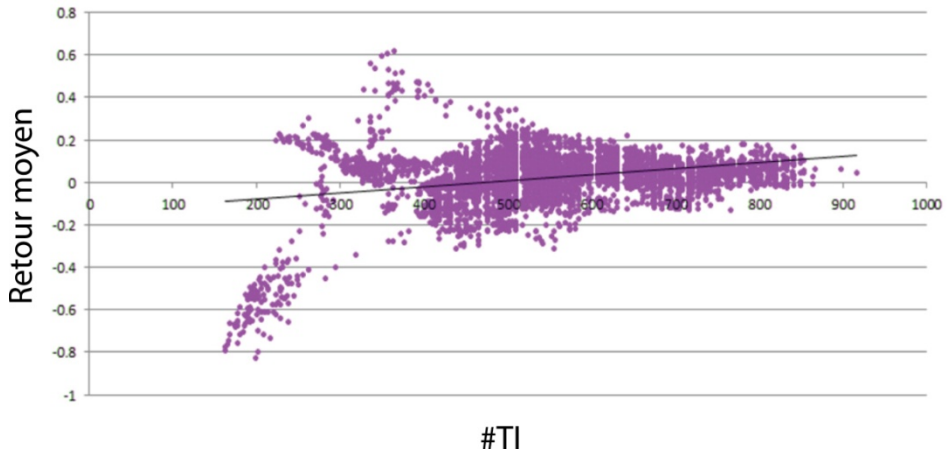


Figure 4.5 Le retour moyen du marché boursier contre le #TI pour plus de 7500 jours consécutifs.

En regardant le graphique ci-dessus, Ben explique :

> If the [#TI] is high, I consider that to be a healthy market. What that means is all the stocks are doing their own thing. […] And my theory is that the more tense the market it is, the more difficult market conditions are, the more the stocks are coupled together. […] You'll notice that the big drops in the market occur when the [#TI] is small.

L'observation des graphiques, comme celui à la Figure 4.5, permet à Ben de développer un modèle mathématique qui estime la volatilité du marché selon la valeur du #TI : si le #TI est petit, on peut prédire qu'il y aura de grandes pertes sur le marché.

Le *calcul des paramètres (f)* implique la détermination des paramètres du modèle qui colle à la situation spécifique à laquelle on s'intéresse. Albert, par exemple, veut étudier le mouvement d'objets voyageant près de la terre (P_1) et de la lune (P_2). Alors, sachant les masses de ces deux corps massifs ($m_1 \approx 5{,}9722 \times 10^{24}$ kg et $m_2 \approx 0{,}07348 \times 10^{24}$ kg), il fixe $M = m_2/(m_1 + m_2) \approx 0{,}01215$ dans (*). Dans cet exemple, la spécification du modèle au contexte est directe ; d'autres cas plus complexes nécessitent toutefois un recours à la programmation. Avant de calculer le risque de crédit sur le marché financier, Adèle a besoin de déterminer les valeurs des 20 paramètres qui apparaissent dans un modèle qu'elle a créé. Pour ce faire, il lui faut utiliser des techniques d'optimisation sur un très grand nombre de données (les produits dérivés de plusieurs compagnies sur le marché à travers le temps). Alain et ses collaborateurs ont aussi à utiliser des méthodes d'optimisation pour calculer avec haute précision les valeurs d'un grand nombre de paramètres (environ 80). Le but dans ce cas est de développer le modèle d'une pompe pour le cœur qui puisse être véritablement construite et utilisée par la suite. Bien que ce modèle paraisse plus fortement ancré dans le réel, le processus de recherche est à peu près le même : pour Alain comme pour Adèle, la programmation est une nécessité dans le développement de leurs modèles pour des raisons *pragmatiques.* Sans l'ordinateur, le grand nombre de calculs numériques à réaliser pour ajuster le modèle au phénomène qu'ils souhaitent décrire ne serait même pas envisageable.

La *validation analytique du modèle (e)* s'approche davantage du travail habituel associé au mathématicien et se fait généralement (ou tout au moins partiellement) sans outil technologique. Une fois le modèle développé, il convient généralement d'en prouver certaines propriétés telles que l'existence, l'unicité et la rigidité des solutions. Cette tâche sert principalement à vérifier que le modèle est bien défini. Mais la connaissance de telles propriétés peut aussi orienter l'accomplissement de la prochaine tâche ; par exemple, si les solutions sont rigides, certains algorithmes pourraient être préférables à d'autres.

Programmation du modèle (3)

Il peut arriver que la programmation du modèle ne soit pas une grande tâche ; cela est d'autant plus vrai si le modèle a déjà été programmé, en totalité ou en partie. Alain, par exemple, croit utile et légitime de profiter de la puissance des programmes commerciaux pour

étudier certains modèles de la mécanique des fluides. Alors, avant même de programmer un modèle mathématique, les chercheurs font souvent de *la recherche sur les outils informatiques existants (a)*. Néanmoins, comme Alice, Alain souligne le fait que les outils existants ne peuvent pas tout faire. Donc, pour avoir plus de contrôle et pour pouvoir faire autre chose que ce qui est prévu par les programmes commerciaux, il doit créer son propre outil.

Si un programme adéquat n'existe pas déjà, la tâche de programmation peut être énorme, comme l'illustre le projet d'Alice. Le processus de *développer un algorithme (b), coder l'algorithme (c)* et *vérifier et valider le programme (d)* peut durer longtemps et résulter en des programmes si complexes qu'ils deviennent difficiles à partager ou même à montrer. Adèle nous a demandé : « Vous voulez voir le code, la programmation ? […] Ça a de milliers de lignes, là. Ce n'est pas intéressant comme tel (rires), je ne sais pas... » Albert a hésité de la même façon à me montrer son code « intimidant ». Les programmes immenses de ces mathématiciens sont des produits importants de leur recherche, mais ils ne peuvent pas prétendre à une transparence équivalente aux productions mathématiques traditionnelles.

Pour *développer les algorithmes (b)* nécessaires à la résolution ou à la simulation des modèles, les mathématiciens commencent en faisant un deuxième niveau de recherche : ils regardent des algorithmes existants, des codes écrits par d'autres et des fonctions préprogrammées disponibles dans les diverses bibliothèques. Bref, en reprenant la typologie de Trouche (2000), les chercheurs considèrent attentivement les *contraintes de commandes* de chaque outil disponible. Ce faisant, ils choisissent les meilleurs outils, déterminent la structure du programme et en précisent les étapes. Même si le processus paraît rigide, il peut exiger beaucoup de créativité. Certains participants témoignent d'une longue réflexion avant d'en arriver à un algorithme efficient et parfois complètement inédit. Ben dit de son propre processus : « Often I have to play, think about it, make models on paper, [...] that's one tough thing about programming is that you have to have a totally clear idea in your mind what you're going to do. You can't just start writing code, it's impossible. »

Mais ce travail sans ordinateur ne consiste pas uniquement à définir les étapes de l'algorithme ; Alice nous en indique une autre fonction importante. « It makes me an extremely smart coder », elle explique, « because I actually think hard about the mathematical issues at hand before I start coding. [...] I write very efficient code; I don't waste resources. I don't need

a supercomputer to run a bad algorithm. I want a smart algorithm on a laptop. » Les mathématiciens appliqués doivent considérer soigneusement les *contraintes internes,* intrinsèquement liées à la machine (Trouche, 2000), et optimiser en conséquence l'efficacité, la stabilité et la précision de l'algorithme. Alice ajoute, par exemple : « If you don't have an insight into how rounding error propagates through your algorithm, you have no idea if the pictures that you're seeing have any bearing whatsoever on what you're trying to compute. So, the pictures don't mean anything. » Autrement dit, sans vraiment comprendre l'algorithme utilisé, un chercheur pourrait mal interpréter les résultats engendrés par son programme.

Selon l'envergure du projet et les gens qui y sont impliqués, *le codage de l'algorithme (c)* pourrait être fait par le chercheur seul ou par son équipe. Mais pour quelques-uns des mathématiciens appliqués, cette tâche est presque toujours déléguée à leurs étudiants de recherche. Dans l'exemple d'Alice, un étudiant est responsable du codage du simulateur. Au sujet de l'écriture de vastes codes, elle commente : « I've done that. I know I can do it. I'm not interested in it. » Elle considère de plus que cela constitue une partie importante de la formation de ses étudiants aux cycles supérieurs. Adèle ajoute une autre raison à cette délégation du travail de codage aux étudiants :

> Ils sont beaucoup plus efficaces que moi pour le faire maintenant. Parce que les langages évoluent, puis parce qu'à un moment donné on finit par rouiller quand on ne le fait pas soi-même. Mais j'ai suffisamment de connaissance des méthodes numériques et de leurs implications pour dire « Bien, ton code n'est pas efficace. »

Si les mathématiciens ne sont pas toujours impliqués au moment du codage, leurs connaissances du problème, des mathématiques et des méthodes leur permettent de participer à l'étape de *vérification et de validation du programme (d).* En somme, cette étape pose la question : le programme donne-t-il vraiment la solution au modèle mathématique ? Pour y répondre, on analyse typiquement le fonctionnement des parties du code en observant les données produites correspondant aux paramètres initiaux pour lesquels le comportement du modèle est connu. Alice considère des cas limites ou des problèmes plus simples. Adèle fait rouler son code sur ses propres bases de données. Mais dans certains cas plus rares, la physique du modèle peut suggérer naturellement d'autres méthodes de vérification. Par exemple, pour le système (*) étudié par Albert, il existe une quantité, l'énergie E du système, qui doit rester constante à chaque moment de la solution :

$$E = \frac{x'^2 + y'^2 + z'^2}{2} - \frac{1}{2}(x^2 + y^2) + \frac{1 - M}{r_1} + \frac{M}{r_2} - \frac{M(1 - M)}{2}.$$

Ainsi, à chaque valeur de la variable temps, on peut calculer la valeur de E. Si la valeur reste constante, on peut avoir plus de confiance dans les résultats du programme ; sinon, on sait qu'il y a une erreur. Enfin, les mathématiciens de notre étude soulignent l'importance de cette étape pour s'assurer que tout va bien avant de sauter aux conclusions.

Recherche des solutions et expérimentation avec le modèle (4)

À l'aide du programme, le chercheur peut étudier le comportement du modèle. Dans certains cas, les mathématiciens cherchent des réponses à des questions très précises. Adèle calcule le risque associé à un ensemble de portefeuilles. Elle procède ensuite à l'analyse des calculs pour évaluer, par exemple, le risque moyen, les plus grands gains et les pires pertes. De même, Ben détermine le #TI chaque semaine et fait des prédictions concernant le marché boursier. Au moment de l'entrevue, il nous disait : « The [#TI] has been dropping for several months now. So my feeling is the market is getting more and more risky. » Le graphique utilisé pour arriver à cette conclusion est montré à la Figure 4.6.

Figure 4.6 Le nombre de titres indépendants (#TI)
pour chaque jour pendant l'année avant le moment de l'entrevue.

Dans l'exemple d'Alice, la mathématicienne commence aussi en calculant les solutions à quelques problèmes spécifiques. Mais éventuellement, son équipe explore les simulations correspondant à un grand nombre de paramètres, sans savoir exactement ce qu'ils cherchent. Comme nous l'avons vu, une telle expérimentation avec un modèle peut mener les chercheurs

à des découvertes. Albert, procédant de manière similaire, peut découvrir des phénomènes célestes remarquables. Par exemple, ses expérimentations numériques le conduisent, avec ses collaborateurs, à l'identification d'un nombre énorme de solutions périodiques de (*). Rappelons-nous que le système (*) modélise le mouvement d'un objet dans l'espace, soumis aux forces d'attraction de deux objets beaucoup plus lourds (la terre et la lune, dans le cas qui intéresse Albert). Ainsi, une solution périodique du système représente une trajectoire fermée où un objet revient à sa position et sa vitesse initiales après un temps fini. Trois solutions périodiques proches de la lune sont montrées à la Figure 4.7. La classification d'une

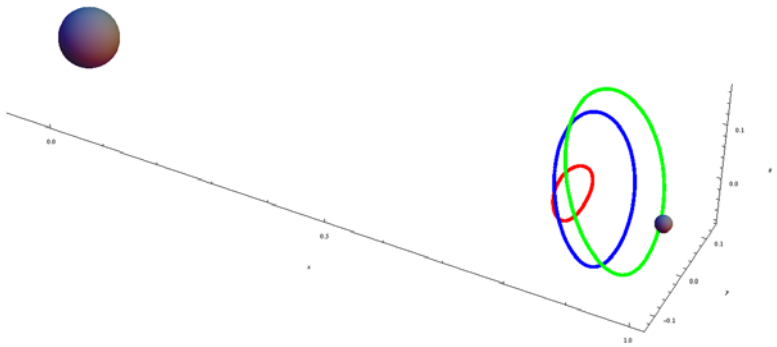


Figure 4.7 Trois solutions périodiques du système (*) avec $M \approx 0,01215$ (P_1 est la terre, P_2 est la lune).

infinité de telles orbites constitue seulement une des contributions dont Albert est très fier. Avec la puissance de son outil informatique, lui et ses collaborateurs calculent plusieurs autres types d'objets dans l'espace. Par exemple, autour d'une solution périodique donnée, ils déterminent l'ensemble des trajectoires issues d'une petite perturbation de l'orbite originale. Cet ensemble, qu'on peut associer à l'objet mathématique de « variété », peut créer des formes magnifiques (voir Figure 4.8 ci-après). Les figures 4.7 et 4.8 sont le fruit d'un programme que nous avons écrit en *Mathematica* pour reproduire les explorations faites par Albert (voir le code en Annexe E). Si cette exploration personnelle nous permet de bien illustrer le potentiel d'un programme informatique, elle nous a aussi offert une meilleure appréciation de la force et de la puissance des travaux d'Albert. Ce qui demande plusieurs minutes de simulation avec le programme que nous avons écrit (en utilisant une méthode numérique simple) prend moins d'une seconde avec les techniques de calcul développées par Albert. Nous ne pourrions certes pas recréer la plupart des résultats obtenus par ce chercheur.

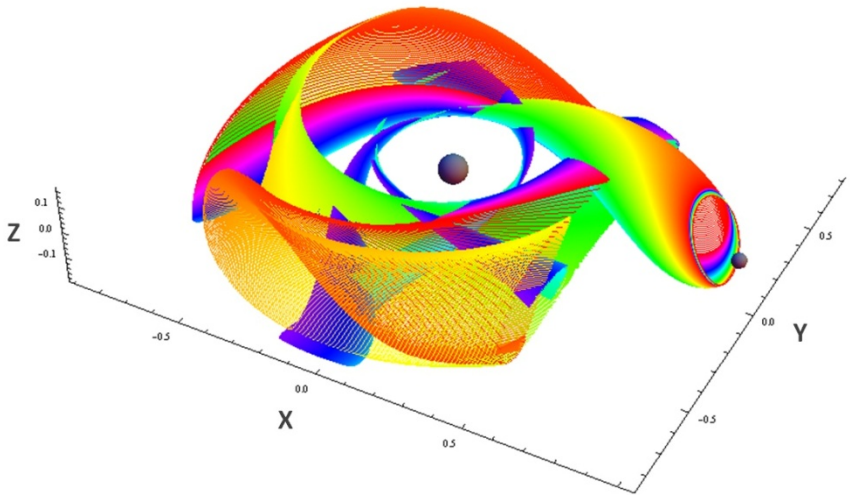


Figure 4.8 Un ensemble de trajectoires qui sont des perturbations d'une orbite périodique proche de la lune.

Notons que l'analyse du comportement d'un modèle ne se fait pas qu'à l'aide de l'ordinateur. Comme l'explique Alice : « you can say a lot of things without actually computing anything. » Barbara en particulier parle de la recherche analytique de plusieurs propriétés de ses modèles en biologie. Elle souligne le travail accompli sans ordinateur qui lui permet d'anticiper le comportement du modèle et, par conséquent, de savoir où commencer ses explorations numériques. Elle dit : « You need to get a good set of parameters, because you can't just choose a set and try it; it's like finding a needle in a haystack. So you would need to do a little bit of analysis to have an idea of where to start. »

Néanmoins, selon tous les mathématiciens œuvrant dans des secteurs appliqués que nous avons rencontrés, une exploration numérique des modèles est éventuellement essentielle au progrès de leurs études. Adèle suggère franchement : « Il n'y aurait pas de projet de recherche sans coder. » Quand ces chercheurs justifient leur investissement dans une activité de programmation afin d'explorer leurs modèles, ils donnent principalement des raisons *pragmatiques*. Plusieurs chercheurs remarquent que leurs modèles mathématiques mènent rarement à des équations qui peuvent être résolues en mode papier-crayon. Alice explique :

> If you're interested in the specific details of the solution, if you're to actually look at what a solution might do, you have to compute because, unless it's very special, you cannot actually write down a formula for a solution, right? […] most models that we're interested in, you cannot tractably write down a solution.

De façon similaire, Ben dit de son travail portant sur le marché boursier : « You could not do this without computers. There will never be an analytic formula for this. It is entirely by simulation. » Dans les mots de Cornu (1992), ce sont alors les capacités de simulation de l'ordinateur qui deviennent indispensables à ce moment d'un projet appliqué.

Les éléments visuels produits par les programmes informatiques donnent aussi une valeur *épistémique* à la programmation faite par ces mathématiciens. C'est essentiellement à cause des capacités graphiques de l'ordinateur qu'Alice et ses collaborateurs peuvent faire la découverte de nouvelles structures dans les fibres des muscles qu'ils étaient en train de simuler. Les graphes utilisés par Ben l'aident à déterminer rapidement le comportement à long terme du #TI et les prédictions correspondantes. Et personne ne peut nier l'impact énorme des images créées par Albert sur la compréhension des phénomènes célestes qu'il découvre. Mais comment valider le comportement complexe observé dans de telles images qui dépassent en puissance ce que l'homme peut dégager de ses observations de la réalité ?

Interprétation et validation des résultats (5)

Il y a deux types de validation mentionnés par les participants de notre étude : la validation mathématique et la validation au regard du phénomène réel. Dans le premier cas, le mathématicien veut prouver les conjectures spécifiques qu'il a pu faire en observant le comportement de son modèle avec un programme informatique. Par exemple, si un mathématicien observe des solutions périodiques pour une certaine famille de paramètres, il peut parfois valider cette observation en développant un argument analytique. Barbara décrit un projet dans lequel elle fait le contraire : elle construit une preuve à l'effet que son modèle a des solutions périodiques, puis elle essaie de vérifier sa preuve en regardant des simulations du modèle. Elle découvre alors la difficulté d'observer la périodicité des solutions dans ses simulations, car celle-ci ne se manifeste que pour des valeurs très petites d'un des paramètres ; cela montre l'importance du premier type de validation.

Le deuxième type de validation est nécessaire pour accomplir ce qui est souvent le but principal d'un projet en mathématiques appliquées : aider à la compréhension du comportement d'un phénomène réel. Pour Alice, « that's when you need to be working with a

scientist. » En particulier, le mathématicien confronte ses résultats avec des données authentiques ou des connaissances empiriques du phénomène réel afin de :

1) déterminer l'applicabilité du modèle : en cerner l'utilité et les limites provenant des simplifications et des choix qui ont été faits. Par exemple, dans le cadre du projet d'Alice, on vérifie si le modèle développé représente bien le comportement de vrais muscles.

2) confirmer de nouvelles prédictions faites à l'aide du modèle. Alice et ses collaborateurs veulent savoir : dans de vrais muscles, est-ce que les fibres produisent les structures que le code a générées ?

Alain est un autre mathématicien appliqué qui insiste sur la nécessité de la validation du modèle avec de véritables expériences. Par exemple, la pompe qu'il modélise a été construite et implantée dans de vrais animaux afin de juger l'applicabilité du modèle. Ces tests aident à déterminer des balises importantes avant d'envisager des implantations de la pompe dans les cœurs d'êtres humains. Alain mentionne aussi l'utilisation d'expériences réelles dans tous ses projets portant sur la mécanique des fluides ; en effet, l'étape de l'expérimentation est si essentielle à son travail qu'il a développé son propre laboratoire physique de dynamique des fluides. Il l'utilise soit pour évaluer ses modèles théoriques, soit pour vérifier de nouvelles solutions obtenues numériquement. Alain conclut enfin : « Dans mon monde à moi, je ne peux pas faire que du numérique ; je fais le numérique et je fais l'expérience pour valider le numérique. »

Remarquons que la validation des résultats au regard du phénomène réel n'est pas toujours facile ou même envisageable. Le projet d'Albert concernant les mouvements célestes en est un bon exemple. Mais cette validation ne revêt pas une grande importance pour Albert lui-même, dont la contribution est la création de nouvelles techniques numériques efficientes :

> Practical people might say "Well it's all very cute, but what importance does it have?" [...] most of all it is to show off, to say, "Look what we can compute and you try it using your techniques. We compute 512 [...] orbits in one run. You try to get ten, okay? Good luck!"

Communication des résultats et préparation pour le futur (6)

Lorsqu'ils atteignent un degré de signification adéquat, les résultats sont partagés avec la communauté mathématique ou scientifique. La partie la plus importante de cette tâche est la description des résultats eux-mêmes, complétée par la présentation des calculs et graphiques qui témoignent des faits saillants. Si un nouveau modèle a été construit, il est évidemment présenté. Mais, selon Adèle, la simple description du modèle ne suffit pas toujours ; elle déclare qu'en ingénierie financière « il faut vraiment quantifier [...] ça prend des exemples avec de vraies fins, de vraies données, de vraies réponses [...] sinon, il y a zéro publications qui sortent de là. » Même Ben, qui choisit de communiquer ses résultats sur un site web plutôt que par articles, fait part d'authentiques prédictions qui pourraient être utiles pour d'autres.

En plus de partager les résultats principaux, la tâche de communication exige souvent de convaincre les lecteurs de la validité de ces résultats. Quelques participants de notre étude parlent de la nécessité de préciser certains aspects de leurs programmes (algorithmes utilisés, efficacité du code, précision des approximations, etc.) et de les comparer aux autres existants ; après tout, la qualité des résultats dépend de la qualité de la programmation qui a été faite. Un nouveau mouvement en recherche encourage par ailleurs à rendre disponibles les programmes élaborés et utilisés, notamment pour rendre les recherches reproductibles. Albert est le seul mathématicien de l'étude qui y contribue toujours. Avec chaque projet, il fournit le programme correspondant. Il a même commencé à mettre en ligne des programmes retravaillés de ses projets passés. Il explique : « It's the programming part now that becomes more important to me. So that other people, those who are interested, can redo the calculations. »

Bien que tous ne contribuent pas à ce mouvement au même degré qu'Albert, tous considèrent important ce partage de programmes, du moins dans certains cas. Barbara, par exemple, justifie sa décision de ne pas partager ses propres programmes en expliquant : « What we do isn't necessarily original programming. I mean it's programming, but it's nothing amazing. It's not the programming that's the focus of the work […] a lot of researchers would be able to program it. » Selon Barbara, la disponibilité des programmes est importante quand la programmation est complexe ou unique. Paradoxalement, c'est la complexité ou le caractère unique d'un programme qui peut aussi servir comme raison de ne pas le partager. Rendre compréhensible des programmes complexes peut demander beaucoup d'efforts. Et si un

mathématicien innove avec son programme en employant des méthodes créatives, il pourrait choisir de ne pas le partager, dans l'avenir immédiat tout au moins, pour garder son avantage compétitif. Il n'est d'ailleurs pas rare qu'un mathématicien continue à utiliser le même programme sur une longue période. Alice, par exemple, peut décider d'adapter son programme à la résolution de plusieurs autres problèmes. De son côté, Ben emploie ses programmes chaque semaine pour analyser le marché boursier et l'on peut deviner le caractère sensible de ces données.

Néanmoins, le partage de programmes, même simples, peut servir deux objectifs importants. Certains participants expliquent que leurs lecteurs font souvent confiance aux processus d'arbitrage et portent peu d'attention à la façon dont les résultats ont été trouvés ; mais comment assurer la qualité et la rigueur de ce qu'on dit avoir fait si les arbitres n'ont pas eux-mêmes accès au code ? Partager n'importe quel programme peut également contribuer au développement de l'ensemble d'outils disponible aux chercheurs, et plus généralement, au développement des connaissances mathématiques.

4.2.2 La résolution de problèmes en mathématiques pures

L'activité de programmation n'a pas exactement le même statut en mathématiques pures qu'en mathématiques appliquées : tandis que l'utilisation de l'ordinateur est essentielle dans la résolution des problèmes appliqués, il existe des problèmes purs qui n'exigent aucun outil informatique. Néanmoins, les expériences diverses des mathématiciens purs de notre étude montrent que la programmation peut être une activité extrêmement fructueuse à certains moments dans le développement des théories abstraites. Pour ouvrir une discussion générale de ces moments, nous commençons à nouveau avec un exemple.

Un exemple illustratif

Olivier est un mathématicien qui travaille en théorie spectrale géométrique. Dans un de ses projets, il cherche à contribuer à l'avancement des connaissances mathématiques en prouvant l'existence d'un certain objet abstrait : une métrique extrémale sur un espace topologique spécifique. Il réduit son problème à l'étude du système d'équations suivant :

$$(**)\qquad \begin{aligned} y_0'' &= (8y_0^2 + 6y_1^2 - 8)y_0\ ; \\ y_1'' &= (8y_0^2 + 6y_1^2 - 7)y_1\ ; \end{aligned}$$

$$y_0(0) = \sqrt{1-c^2}\ ;\ \ y_0'(0) = 0\ ;$$

$$y_1(0) = 0\ ;\ \ y_1'(0) = 2c.$$

Avant d'être capable d'étudier ce système, Olivier a besoin de trouver la valeur du paramètre c, qui apparaît dans les conditions initiales, selon deux contraintes spécifiées : (1) c se trouve dans l'intervalle $(0, 1)$; et (2) c doit mener à des solutions périodiques qui ont exactement deux zéros dans la période.

Olivier explique qu'à l'époque, « ce n'était pas du tout clair comment trouver la bonne valeur du paramètre. Et donc, […] j'ai programmé le système […] *Mathematica* permet de trouver les graphes des solutions. Et j'ai commencé à jouer avec le paramètre […] pour voir ce qui se passe. » Même si Olivier n'était pas en mesure de partager au moment de l'entrevue le programme qu'il avait utilisé pour effectuer son exploration, nous avons pu créer un programme en *Mathematica* qui nous permette de faire les mêmes observations que lui :

```
c = 0.1;

s = NDSolve[{y0''[x] == (8*(y0[x])^2+6*(y1[x])^2-8)*y0[x],
y1''[x] == (8*(y0[x])^2+6*(y1[x])^2-7)*y1[x], y0[0] == Sqrt[1-c^2],
y0'[0] == 0,  y1[0] == 0, y1'[0] == 2c}, {y0, y1}, {x, 0, 30}];

Plot[Evaluate[{y0[x],y1[x]}/.s], {x,0,30}, PlotStyle->Automatic]
```

Un tel programme ne relève pas d'une programmation très complexe. Il nous a suffi de repérer dans la documentation la fonction préprogrammée « NDSolve » (qui résout de façon numérique un système d'équations différentielles ordinaires), d'y préciser le système d'équations et de la faire suivre d'un « Plot » des deux fonctions pour avoir une représentation graphique de la solution. Mais la simplicité du programme ne dit rien sur son utilité. Il nous permet de générer très rapidement les graphiques des fonctions-solutions pour différentes valeurs de c et d'en déduire que la valeur qui répond aux exigences se trouve près de 0,6. Remarquons, par exemple, que parmi tous les graphiques montrés à la Figure 4.9, seul le

quatrième présente des solutions qui pourraient être périodiques avec deux zéros dans la période (la deuxième contrainte mentionnée ci-avant). Par contre, cette contrainte ne semble pas être satisfaite si c s'éloigne de 0,6 : même si les autres graphiques ci-dessous pourraient manifester une certaine périodicité, ils auraient tous trop de zéros dans la période. Olivier arrive à cette découverte en examinant des graphiques similaires.

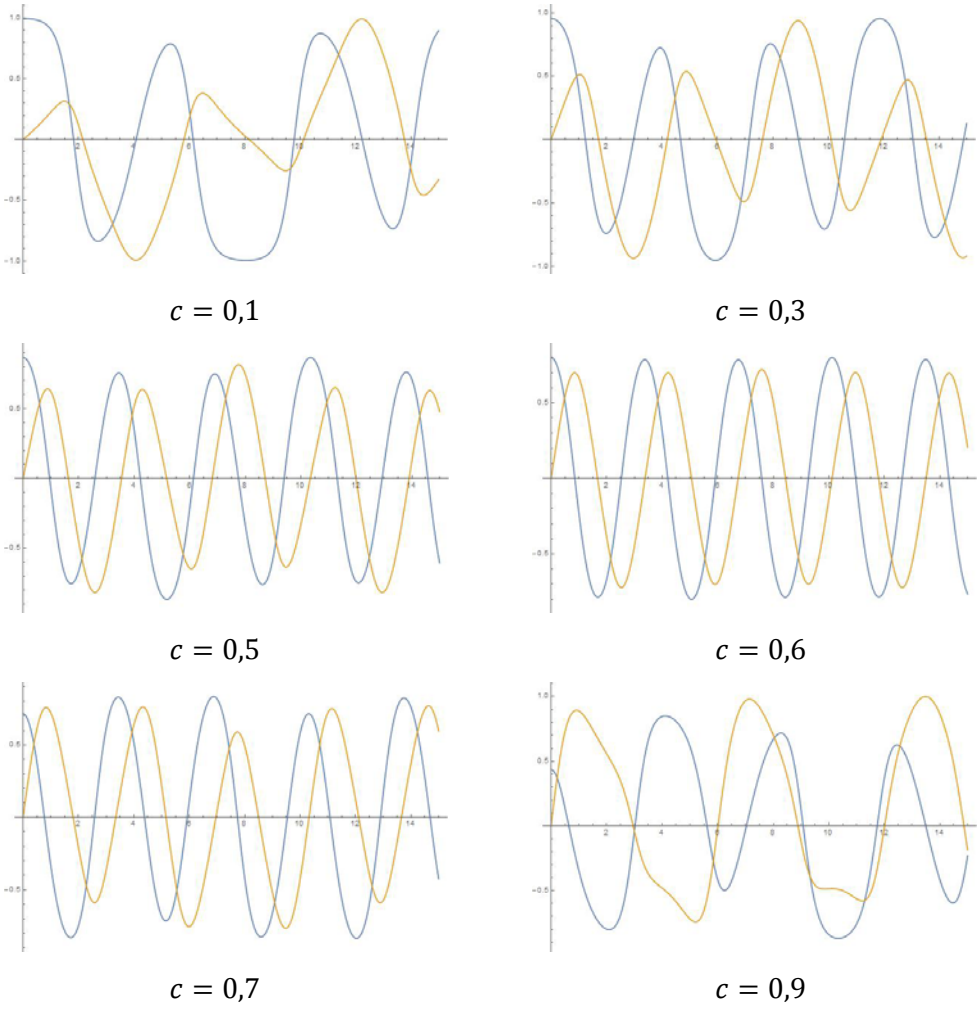


Figure 4.9 Les solutions du système (**) pour différentes valeurs de c.

Des expériences de ce type, avec *Mathematica* ou un autre logiciel de calcul numérique, ne fournissent qu'une valeur approchée de la valeur cherchée. C'est seulement en ayant fait les expériences nous-mêmes que nous avons pu apprécier le problème de précision qu'Olivier a évoqué pendant l'entrevue. Par exemple, à cause des limites de résolution de l'écran, ce qui résulte d'une *contrainte interne* (en employant encore la typologie de Trouche,

2000), il est très difficile de juger si les solutions satisfont à la condition (2) à partir de la quatrième décimale de c. La représentation sur l'intervalle [0, 5] des courbes des solutions pour $c = 0{,}6120$ et $c = 0{,}6125$ (Figure 4.10) permet d'illustrer la difficulté de trancher par simple observation en faveur de l'une ou l'autre de ces valeurs ; les deux graphiques semblent montrer des solutions périodiques qui ont deux zéros dans la période.

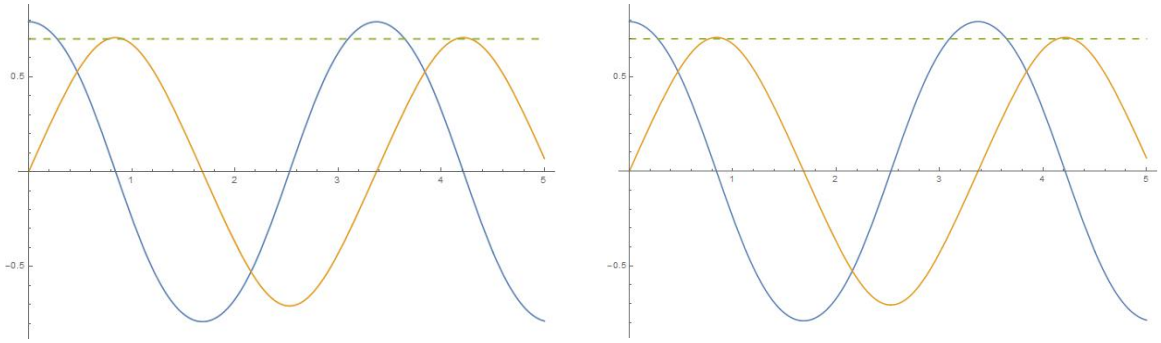


Figure 4.10 Les solutions de (**)
pour $c = 0,6120$ (à gauche) et $c = 0,6125$ (à droite).

Pour Olivier, comme dans bien des projets en mathématiques pures, une approximation décimale n'est pas suffisante. Il lui faut trouver « la vraie valeur » de c. Que fait-il alors ? Il décrit ses actions de la façon suivante :

> J'ai commencé à réfléchir, donc : qu'est-ce que ça peut être ? Et j'ai commencé à faire des opérations différentes. Donc j'ai essayé de voir est-ce qu'il y a un lien avec le nombre π [...] Et, finalement, j'ai réalisé que ce nombre c'est juste $\sqrt{3/8}$. C'était juste comme une révélation. Parce que ce nombre irrationnel était vraiment extrêmement proche de la valeur que j'ai trouvée numériquement.

Olivier procède par un processus systématique d'essai et erreur en cherchant à reconnaître un nombre relativement simple. Il utilise *Mathematica* encore, mais cette fois-ci simplement pour calculer la forme décimale de chacun des candidats qu'il envisage pour ensuite la comparer à l'approximation de c qu'il a déjà trouvée. Quand il finit par tomber sur un nombre très proche de son approximation, il a finalement une bonne conjecture : $c = \sqrt{3/8}$.

Avant de retenir $c = \sqrt{3/8}$ et de continuer avec la résolution du problème plus grand dans lequel s'inscrivait l'étude du système (**), Olivier doit être certain de sa conjecture. Pour se convaincre, il substitue $c = \sqrt{3/8}$ dans les conditions initiales du système, trouve les

solutions analytiques correspondantes et utilise des techniques d'analyse pour démontrer que ces solutions sont vraiment périodiques avec deux zéros dans la période. Autrement dit, il travaille à la main avec papier et crayon pour construire une preuve formelle de sa conjecture. Une fois la preuve rédigée, Olivier n'a plus besoin des matériaux qu'il a utilisés pendant ses explorations, y compris de son petit programme informatique ; c'est pourquoi il ne le garde pas et ne pourrait donc pas le partager avec nous pendant l'entrevue. Il explique : « quand j'ai été capable de la faire [(la preuve analytique)], tout ce que j'ai fait en tant que programmation est devenu inutile [...] Mais je n'aurais jamais été capable d'arriver à ce point sans les expériences que j'ai faites. » Dans l'article communiquant ses résultats, Olivier mentionne en une phrase l'utilisation d'expériences numériques pour cibler la valeur de c et enchaîne avec la preuve analytique sur plusieurs pages qui démontre que $c = \sqrt{3/8}$.

Les tâches génériques du processus de recherche en mathématiques pures

Tout comme Alice, Olivier note l'intemporalité du processus de recherche auquel il s'associe : « Je ne pense pas qu'il ait changé beaucoup sur des centaines d'années. Donc c'est vraiment une science très conservatrice de ce point de vue. » Dans son exemple, nous identifions la liste suivante de tâches génériques, où l'exploration séculaire s'appuie ici sur une mise à contribution de la programmation :

1) choix et délimitation d'un problème mathématique ;
2) création de programmes informatiques ;
    a) développement d'un algorithme ;
    b) codage de l'algorithme ;
    c) vérification et validation du programme ;
3) exploration ;
    a) observation d'objets mathématiques ;
    b) formulation des conjectures ;
    c) vérification des conjectures ;
4) preuve ;
5) communication des résultats et préparation pour le futur.

Une fois de plus, notons que ces tâches ne sont pas nécessairement vécues de façon égale, distincte ou linéaire dans chaque projet. Mais elles aident à situer le rôle de la programmation dans un processus de recherche en mathématiques pures. D'autres exemples décrits par nos participants nous permettent d'ajouter les détails nécessaires pour mieux comprendre chacune de ces tâches, l'implication des programmes informatiques dans leur réalisation et donc la place et l'apport de la programmation dans les pratiques de recherche en mathématiques pures. Ceci est le sujet de ce qui suit.

La description des tâches et de la place de la programmation

Choix et délimitation d'un problème mathématique (1)

Pour les mathématiciens purs, le choix d'un problème semble être une tâche personnelle et délicate. Olivier explique qu'en mathématiques fondamentales :

> Le défi le plus sérieux, si vous voulez, est de trouver un bon problème. […] c'était un grand mathématicien [...] qui a dit que : On peut voir les mathématiques comme une noix. [...] vous avez tout ce qui est à l'extérieur de la noix ; ce sont les choses qui sont inaccessibles. Et vous avez le noyau de la noix, qui est fait des choses triviales. Et donc, c'est juste dans la frontière où sont les choses qui sont en même temps accessibles et non triviales. [...] c'est exactement là où les mathématiciens font leur recherche.

Contrairement aux mathématiciens appliqués, les mathématiciens de ce sous-groupe définissent eux-mêmes leurs problèmes selon leurs intérêts, mais aussi selon les besoins de la communauté mathématique. Ben, qui semble se situer entre les mathématiques appliquées et les mathématiques pures, suggère : « When you're a mathematician, […] you can see the questions that just somehow pop out as these are valuable, we need to know this. *We* as mathematicians, not the world. » Après avoir identifié une question assez importante d'un point de vue mathématique, le chercheur divise souvent son travail en sous-questions et utilise des outils théoriques pour les reformuler ; cette délimitation du problème, selon Olivier, peut aussi être très exigeante.

Dans la résolution de problèmes en mathématiques pures, l'utilisation de programmes informatiques n'est pas toujours requise. En fait, plusieurs mathématiciens ne créent aucun programme informatique pour résoudre leurs problèmes. Parmi les mathématiciens que nous

avons rencontrés, Philippe et Nathan déclarent ne jamais programmer (ou presque) dans leur propre recherche. Même Omar, qui a décrit plusieurs moments où il écrit des programmes informatiques, explique : « People doing numerical analysis or statistics on big sets of data, it's useless unless they do programming on a weekly, on a daily basis. In my case, I can be a month or two without using *Mathematica*. » Pierre, un des mathématiciens très purs, résume : « En général, il n'y a pas un outil de recherche qu'on puisse utiliser pour tous les problèmes. Donc, si on change le problème, on va changer peut-être aussi les outils qu'on doit utiliser. »

Création de programmes informatiques (2)

Si un mathématicien n'a pas besoin d'un programme informatique pour résoudre ses problèmes, cette deuxième tâche n'est pas nécessaire. Mais plusieurs des mathématiciens purs rencontrés ont parlé des moments où ils programment pour créer leurs propres outils d'exploration. Rappelons que pour les mathématiciens appliqués, la programmation requise dans leurs projets est souvent extrêmement complexe ; comme nous l'avons vu dans l'exemple d'Olivier, ce n'est pas toujours le cas pour les mathématiciens purs. Nous pouvons dire que la programmation faite par Olivier est très « spontanée » : au moment où il reconnaît le besoin d'un outil informatique, le mathématicien va immédiatement à l'ordinateur ; il développe un algorithme et le code en même temps ; l'étape de vérification et de validation est presque inexistante ; et une fois que les résultats nécessaires sont obtenus, le programme n'est pas gardé. Cette programmation « spontanée » correspond typiquement à l'utilisation des fonctions préprogrammées connues dans un logiciel de calcul symbolique comme *Mathematica, Maple* ou *MATLAB*.

Certains mathématiciens purs rencontrés vont plus loin dans l'intégration de la programmation à leur recherche. En effet, les techniques de programmation et les programmes de ces chercheurs varient énormément. À un niveau supérieur aux programmes spontanés serait ce que nous appelons une programmation « planifiée et partageable. » Si on les compare au programme d'Olivier, ces programmes sont passablement plus compliqués : ils requièrent souvent une réflexion sans ordinateur, une écriture du code en étapes et un processus de vérification et de validation plus important. Les mathématiciens purs qui se situent à ce niveau mentionnent une vérification par le codage du même programme par plusieurs mathématiciens

et même la validation de certains résultats en les comparant aux mêmes calculs refaits à la main. Les programmes dans ce cas sont typiquement gardés plus longtemps que le programme d'Olivier. Ceci étant dit, ils demeurent facilement explicables à autrui. Omar, par exemple, nous a montré un tel programme pendant l'entrevue.

Deux mathématiciens purs ont hésité d'une façon similaire à Alice et à Adèle quand nous leur avons demandé de partager des morceaux de code durant l'entrevue. Tout comme les mathématiciens appliqués, leur programmation est souvent « complexe » : ils créent de longs programmes avec des algorithmes sophistiqués qui peuvent exiger une prise en considération sérieuse des *contraintes internes* (p.e., la limitation de la mémoire ; Trouche, 2000) pour optimiser notamment l'efficacité du programme. Mais ces programmes n'utilisent pas du numérique aussi souvent qu'en mathématiques appliquées. Sur ce sujet, Pierre réfléchit :

> Je trouve que quand on parle de la programmation [...] souvent on parle d'analyse numérique. « Number Crunching ». Le type de calcul que je fais, ce n'est pas du « Number Crunching », pas du tout. [...] je ne fais pas d'approximations. Quand je calcule le déterminant d'une matrice, je veux la valeur exacte.

Pierre est un mathématicien pur qui crée des programmes immenses pour étudier des problèmes en combinatoire algébrique ; par exemple, un de ses programmes compte plus de 4000 lignes de code. Si ce nombre ne rivalise pas avec les quelque 10 000 lignes des programmes évoqués par nos mathématiciens appliqués, les programmes de Pierre sont souvent loin d'être spontanés. Dans certains cas, ils constituent même des contributions importantes à sa communauté de recherche.

L'autre mathématicien qui hésitait à partager son code avec nous pendant l'entrevue est l'un des quatre probabilistes. Norman, comme plusieurs autres participants, a fait du travail en mathématiques pures et en mathématiques appliquées, et donc, il a utilisé des programmes informatiques dans des contextes très différents. Un de ses projets qui relève des mathématiques pures illustre bien comment ce secteur peut bénéficier de programmes complexes, dans les différentes tâches qu'on y réalise. Pour mieux apprécier cet exemple, nous prenons le temps de présenter les idées fondamentales sur lesquelles il s'appuie.

Une permutation de longueur n est un réarrangement des entiers naturels $\{1, 2, \dots, n\}$. On peut alors la représenter comme une chaîne $\alpha = \alpha_1\alpha_2 \dots \alpha_n$ où chaque α_i est un élément de

l'ensemble $\{1, 2, \ldots, n\}$ et $\alpha_i \neq \alpha_j$ quand $i \neq j$. Par exemple, $\alpha = 624531$ est une permutation de longueur 6. Pour visualiser une permutation α, on peut la voir comme une fonction qui envoie i à α_i. Le graphe associé à la permutation $\alpha = 624531$ est montré à la Figure 4.11.

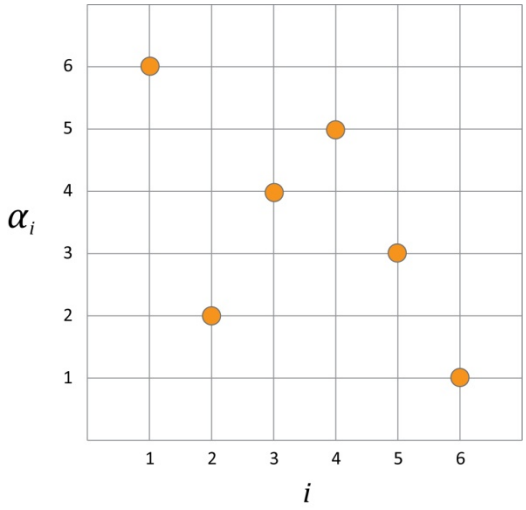


Figure 4.11 Graphe de la permutation $\alpha = 624531$.

Considérons maintenant une autre permutation : $\beta = 231$. On dit qu'*une permutation contient la structure* β si elle contient une sous-suite (pas nécessairement consécutive) dont les chiffres ont le même ordre relatif que 231. Par exemple, la permutation $\alpha = 624531$ contient la structure 231 parce qu'elle a la sous-suite 451 : 1 est le plus petit chiffre, 5 est le plus grand chiffre, et 4 se trouve entre les deux. Mais nous aurions pu prendre également les sous-suites 241, 251, 231 ou 453. Si une permutation ne contient pas une structure donnée, nous disons qu'elle *évite* cette structure. Par exemple, $\alpha = 624531$ évite la structure 1234.

Dans la communauté de recherche à laquelle Norman appartient, un problème commun est celui de trouver le nombre de permutations de longueur n qui évitent une structure donnée. En général, les mathématiciens ont prouvé que, pour une structure de longueur fixée, ce nombre croît à peu près de façon exponentielle avec la longueur n des permutations considérées. Notons par p_n le nombre de permutations de longueur n qui évitent la structure S à laquelle Norman s'intéresse. Pour aider à la recherche d'une forme fonctionnelle pour p_n, Norman et son collaborateur développent un programme complexe qui serait beaucoup plus difficile à reproduire que celui d'Olivier. La définition du meilleur algorithme exige l'évaluation de plusieurs options. Par exemple, une approche serait de faire des calculs exacts

de p_n pour autant de valeurs de n que possible et puis de conjecturer une forme pour p_n. Norman nous a expliqué que cette approche est inefficace : au moment où il menait son projet, on n'était en mesure de calculer les valeurs exactes de p_n que pour $n \leq 25$. Les mathématiciens décident alors d'utiliser une autre approche : celle de trouver des valeurs approchées pour p_n par une méthode probabiliste. Comme c'est souvent le cas avec les projets appliqués, le codage de la méthode a été délégué à un étudiant aux cycles supérieurs et Norman est intervenu au moment de la vérification et de la validation. Cette étape était particulièrement importante ; pour obtenir de bonnes approximations, les chercheurs avaient besoin d'un algorithme très efficace.

Exploration : Observations, formulation des conjectures et vérification (3)

Afin de trouver des solutions spécifiques ou de faire de découvertes, le mathématicien pur entre souvent dans un mode exploratoire. C'est là où il pourrait décider de faire appel aux programmes, spontanés, planifiés ou complexes, pour l'aider à accomplir différentes tâches. En employant les mots de Cornu (1992), il utilise l'ordinateur pour expérimenter.

La première tâche rencontrée par un mathématicien en mode d'exploration est souvent *l'observation du comportement des objets mathématique*s *(a)* à l'étude. Le mathématicien peut choisir de procéder par observations systématiques pour restreindre l'espace de recherche ; il sait ce qu'il cherche et il essaie de s'y diriger. C'est certainement le cas dans l'exemple d'Olivier quand il essaie de réduire l'intervalle où se trouve la valeur de c. C'est aussi le cas de Norman lorsqu'il essaie d'identifier la valeur de p_n. Mais il peut arriver qu'un mathématicien ne sache pas exactement ce qu'il cherche au moment de commencer ses observations ; il essaie plutôt de tomber sur des régularités ou de nouvelles propriétés. Par exemple, en faisant les premiers calculs de p_n, Norman et son collaborateur remarquent que leur programme roule trop lentement pour leur permettre de s'approcher suffisamment d'une solution au problème. Norman suggère alors : « Let's just look at what these permutations look like […] maybe there's some pattern that you can spot in here, some kind of structure that will show out. […] it might give us an insight into how to construct this. » Norman avoue qu'il n'était pas convaincu de trouver ainsi quelque chose d'importance ; après tout, si on choisit au hasard une permutation qui évite la structure S, on ne devrait trouver qu'un nuage de points aléatoires.

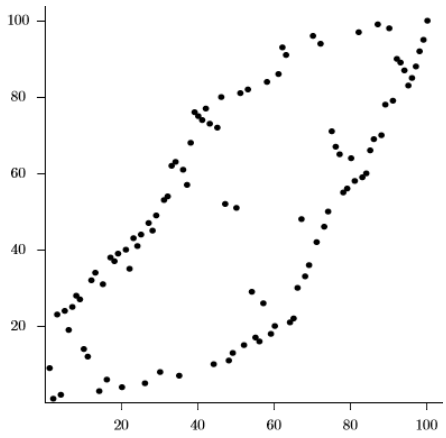


Figure 4.12 Le graphe d'une permutation de longueur 100 choisie au hasard qui évite la structure S. La représentation de la permutation est comme celle de la Figure 4.11.

Mais en voyant des images comme celle à la Figure 4.12, Norman change complètement son point de vue : « I don't understand what it is. But clearly there's something going on there. »

Évidemment, la création de programmes informatiques pour faire des observations a une valeur *pragmatique* : on peut effectuer des calculs ou dessiner des images plus rapidement avec l'ordinateur qu'à la main. En effet, les observations faites par Norman auraient été impossibles sans son programme informatique. Ceci étant dit, les justifications que donnent les mathématiciens pour faire de la programmation à ce moment tendent à avoir une nature *épistémique*. Pierre résume les raisons fournies par plusieurs participants en citant le mathématicien Richard Hamming : « The purpose of computing is insight, not numbers. » En regardant le graphique de la Figure 4.12, on peut dégager immédiatement les régularités dans la structure des permutations qui évitent S. Norman ajoute : « It conveys a totally different impression that a list of matrix entries would not convey. » Olivier profite des capacités graphiques de l'ordinateur de façon similaire ; les graphes des solutions du système d'équations (**) lui permettent de voir facilement le comportement qu'elles manifestent.

À un certain moment pendant l'exploration d'objets mathématiques, les mathématiciens dégagent suffisamment d'observations pour *formuler des conjectures (b)*, c'est-à-dire pour créer des énoncés mathématiques incluant des hypothèses et conclusions qui paraissent vrais mais pourraient s'avérer faux. Dans l'exemple d'Olivier, sa conjecture finale, qu'il pourra éventuellement prouver, est : pour $c = \sqrt{3/8}$, les solutions du système (**) sont

périodiques avec deux zéros par période. Les conjectures de Norman sont un peu plus complexes : par exemple, pour les permutations qui évitent la structure S, on ne trouve aucun point dans les coins supérieur gauche et inférieur droit du nuage de points associé.

Avant de s'attaquer à la preuve d'une conjecture, les mathématiciens veulent souvent se convaincre de sa vérité en *cherchant à la vérifier pour d'autres cas (c)*. L'ampleur de cette tâche varie selon la complexité de la conjecture. Par exemple, Olivier teste un candidat pour c en faisant un seul calcul simple. Par contre, Norman vérifie sa conjecture en regardant un grand nombre d'exemples de permutations qui évitent la structure S (voir la Figure 4.13 ci-dessous). Ce qui différencie la conjecture de Norman de celle d'Olivier est que, dans le premier cas, l'énoncé est lié à une infinité d'objets.

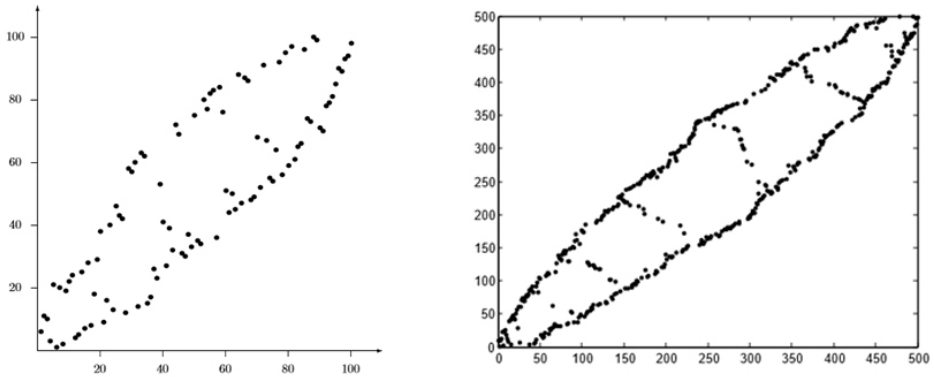


Figure 4.13 Les graphes d'autres permutations choisies au hasard qui évitent la structure S ; $n = 100$ (à gauche) et $n = 500$ (à droite).

Rappelons à nouveau que pour bien des projets en mathématiques pures, toute cette phase d'exploration peut se faire seulement avec du papier, un crayon et une poubelle. Omar, par exemple, décrit un problème où il ne va à l'ordinateur qu'au moment où il croit avoir en mains une solution potentielle. Avant de mettre son énergie à prouver sa conjecture, il écrit un programme afin de la vérifier sur un sous-ensemble de cas possibles. Il justifie une telle action en suggérant : « Before starting to prove something, you'd better know it's true beyond a doubt. » Dans les cas les plus simples, il ajoute même un second niveau de vérification en refaisant les mêmes calculs à la main. Mais à partir d'un certain point, la puissance de l'ordinateur devient vraiment essentielle ; en témoigne l'exécution sur onze heures de son programme pour vérifier la validité de sa conjecture.

Encore une fois, nous voyons que l'utilisation d'un programme informatique peut avoir une grande valeur *pragmatique* ; pour Omar, comme pour Norman, la vérification dans certains cas est simplement impossible sans l'ordinateur. Nous devons aussi mentionner les avantages du caractère adaptable des programmes : une fois qu'on a programmé un outil, on peut simplement changer des aspects pour faire autant de tests que l'on veut. Pierre explique : « Une chose qui est plutôt remarquable avec un ordinateur, c'est que c'est vraiment facile de tester une certaine idée, même si cette idée est idiote. » Du côté *épistémique*, l'observation d'un seul contre-exemple suffit à montrer qu'une conjecture est fausse, tandis que des tests dans des cas variés peuvent suggérer qu'une conjecture est vraie, au point de mériter un essai pour la prouver. Nathan justifie la seule fois qu'il crée son propre outil en expliquant : « La programmation en ce cas-là m'a servi à me donner confiance en mon intuition, pour travailler plus fort sur l'aspect théorique, pour écrire la preuve. »

Il est probable qu'un mathématicien ne trouve pas la bonne conjecture la première fois qu'il essaie de le faire ; il développe une conjecture, la teste, la raffine ou la change complètement, teste sa nouvelle conjecture, etc. Le mathématicien peut ainsi rester dans ce cycle de faire des observations, formuler des conjectures et les vérifier, parfois pendant très longtemps. À partir du moment où il juge avoir une assez forte conviction pour une idée, le mathématicien peut sortir du cycle d'exploration pour compléter la prochaine tâche : l'écriture d'une preuve formelle.

Preuve (4)

La rédaction d'une preuve est souvent le but ultime des projets en mathématiques pures. Pierre explique : « À la fin, quand j'écris un papier, je veux vraiment une démonstration [...] rigoureuse de ce que je suis en train de faire. Sinon, on dit "Bon, ça c'est quelque chose que j'ai remarqué. Peut-être que c'est vrai, peut-être pas. Peut-être que c'est une conjecture." » Tous les exemples que nous avons mentionnés finissent avec cette validation suprême des résultats. Par exemple, Norman démontre enfin qu'il existe des régions avec une haute probabilité d'être vide dans les graphiques des permutations qui évitent S.

Mais la construction d'une preuve analytique à la main n'est pas toujours facile ou même possible au moment du projet, ce qui peut mener des mathématiciens à chercher d'autres

méthodes de validation. Olivier, par exemple, décrit un autre projet dans lequel un calcul rapide en *Mathematica,* pour plusieurs valeurs de w, lui donne l'égalité suivante :

$$\sum_{j=0}^{w} \frac{1}{(w-j)!\,(j+1)!\,(2j+1)} \sum_{k=0}^{j} \frac{(-1)^k k^{2j+2n}}{(j-k)!\,(j+k)!} = 0\,.$$

Le problème est que le mathématicien ne sait pas du tout comment l'expliquer. Tout juste après avoir rencontré ce défi, il a la chance de parler avec un expert en identités combinatoires qui lui donne une idée pour prouver l'égalité. L'élément intéressant de cette anecdote est que cet expert a développé un programme spécial pour construire les preuves d'identités combinatoires. Olivier en décrit l'utilisation : « Vous mettez dans l'ordinateur votre identité et l'ordinateur vous donne la preuve. Ce n'est pas juste le résultat, mais la preuve ! » Si Olivier n'a pas utilisé ce programme dans ce cas, il en approuve entièrement la légitimité : « Je suis 100% pour l'utilisation de ça. Pour faciliter notre vie. » L'utilisation de l'ordinateur pour démontrer (une autre capacité soulignée par Cornu en 1992) aurait donc une grande valeur *pragmatique*.

Paul décrit un autre projet qui illustre l'apport *pragmatique* des programmes informatiques conçus pour donner toutes les étapes d'une manipulation algébrique conduisant au résultat d'un calcul. Dans ce projet, lui et ses collaborateurs effectuent un calcul symbolique à la main sur plusieurs pages de travail. Ce calcul est si complexe que les mathématiciens n'en sont plus sûrs à la fin. Un des chercheurs développe alors un programme en *Python* qui peut effectuer le même calcul. Paul dit du programme :

> You could give it the input that you wanted, and then it would just move the symbols around, which is basically the result of doing this very long calculation […] then it actually gave the output in *LaTeX* code, so you [...] see the equations all formatted and everything. It was kind of amazing!

Le programme ainsi créé montre plus que le résultat d'un calcul : il montre aussi toutes les étapes prises pour y arriver. Ainsi, en investissant dans la création d'un nouveau programme, plus général, plus abstrait, d'une portée plus grande que le calcul initial, les chercheurs se dotent d'un outil qui pourra prendre en charge plusieurs de leurs calculs futurs.

Évidemment, ce ne sont pas toutes les preuves qui sont réalisables complètement par ordinateur. Et il arrive qu'un mathématicien ne puisse pas prouver lui-même une conjecture donnée. Que faire alors ? Parfois un mathématicien choisit d'accumuler des vérifications pour rendre sa conjecture plus convaincante. En effet, dans certains cas, un mathématicien peut considérer une conjecture forte comme un résultat important, méritant d'être publié. Par exemple, Norman et ses collaborateurs sont toujours conscients de la difficulté de déterminer le nombre exact de permutations qui évitent la structure S. Norman suggère même : « It seems unlikely that anyone will know the exact value of $[p_n]$ in the near future. » Alors, les mathématiciens essaient de trouver une bonne approximation à cette valeur et travaillent fort pour convaincre de la précision de leur approximation. Enfin, que ce soit ou non le sujet principal d'un article, il n'est pas rare qu'un mathématicien présente à la communauté des conjectures qui pourraient inspirées des études futures.

Communication des résultats et préparation pour le futur (5)

Éventuellement, les mathématiciens informent la communauté de leurs résultats. Cette communication comprend typiquement une description du problème, une formulation des résultats démontrés, leur preuve, des conjectures qu'il reste à prouver et des exemples pour souligner les éléments importants ou pour convaincre de l'intérêt d'une conjecture.

Plusieurs mathématiciens mentionnent le potentiel de la programmation à ce moment pour produire des figures marquantes. Parfois ces images sont inspirées d'explorations faites par le mathématicien. D'autres fois, elles ne sont développées qu'au moment de la communication. Paul, par exemple, décrit un projet dans lequel il ne programme pas du tout ; mais en écrivant son article, il reconnaît la valeur d'y ajouter des éléments visuels : « There's just so many beautiful images out there that are so revealing and evocative of the real mathematics, and that help you to think about what that object is. » Un programme informatique est donc écrit par un collègue de Paul spécifiquement pour créer les images que le chercheur a en tête. S'il y a des raisons *pragmatiques* pour programmer dans ces cas, l'utilisation des programmes est largement justifiée par son apport *épistémique.*

Mais que peut-on dire de la visibilité des programmes utilisés dans la découverte des résultats ? Si un mathématicien finit par trouver une preuve analytique des résultats, il n'a pas

à parler de l'étape de programmation. Olivier explique : « Si quelqu'un annonce un résultat, il faut que les autres puissent le répéter. Et donc, si la preuve est analytique, la preuve est écrite, donc un chercheur particulier peut la lire et vérifier que tout fonctionne bien. » Alors, l'activité de programmation demeure complètement cachée dans ce cas. Omar, par exemple, mentionne n'avoir appris que récemment que quelques pionniers de son domaine, des mathématiciens purs, ont souvent programmé pour trouver le bon chemin même s'ils n'en ont jamais parlé dans leurs articles. Ce mathématicien envisage la visibilité de sa propre programmation de la façon suivante :

> I might provide an example, which I might have computed with the computer. But the example would be provided because it has a pedagogical value, not because I want to inform the reader that it was useful to use the computer. The computer is a tool. I've never said "I've used two pads of paper to prove this thing." People know that I might use paper, a pen, and a computer. The important result is the theorem with the proof.

Si un mathématicien n'a pas trouvé de preuve analytique pour une conjecture qu'il a découverte et dont il souhaite convaincre la communauté, l'histoire est complètement différente : nos participants s'entendent pour affirmer qu'on doit parler du programme et peut-être même le rendre utilisable par la communauté. Les mathématiciens de notre étude qui développent des conjectures convaincantes (p.e., Norman avec sa conjecture de p_n) parlent dans leurs articles des caractéristiques de leurs programmes telles que l'algorithme, le langage, la convergence, la précision et l'efficacité. Mais ils s'interrogent sur la meilleure façon de partager les programmes pour contribuer au mouvement vers des recherches reproductibles. Paul, par exemple, argue qu'il n'est pas possible de mettre les programmes en annexe des articles puisque les revues n'accepteraient jamais autant de pages supplémentaires. De plus, il se préoccupe de la permanence de l'accès si les chercheurs se limitent à la mise en ligne du programme sur leurs sites web personnels. Il conclut que le site de partage de documents scientifiques arXiv.org pourrait être un lieu de dépôt possible. Mais, comme le mentionnent les mathématiciens appliqués, il y a toujours l'obstacle de l'effort requis pour préparer un programme pour dissémination. Paul explique : « If it's private, then maybe you won't pay the same attention to the documentation of the program. Right? Or to writing some sort of an introduction so that somebody else who picks it up can figure out what it's all about. So, there's a little bit of overhead in that respect. » Néanmoins, il est possible que ce travail vaille

la peine ; après tout, comme nous l'avons déjà mentionné, la disponibilité des programmes peut aussi faciliter des recherches futures.

En général, le partage d'outils semble être fait moins fréquemment par les mathématiciens purs. Si la programmation est « spontanée », les programmes ne sont même pas conservés ; et lorsque les programmes sont ajoutés à la bibliothèque personnelle du mathématicien, c'est parfois leur destination finale. Mais il arrive que des mathématiciens décident de rendre leurs programmes disponibles pour la communauté. Par exemple, le programme développé par Paul et ses collaborateurs pour refaire un calcul symbolique complexe est maintenant disponible sur son site web personnel. Pierre est le seul mathématicien pur qui contribue assez régulièrement au développement d'une bibliothèque ouverte des programmes. Mais, même pour lui, il y a des conditions à considérer avant de déposer un programme en ligne : celui-ci doit être suffisamment intéressant et utile pour valoir le travail supplémentaire requis par le processus de revue de code au sein de la communauté de développement concernée (*Sage*, dans son cas). Pierre explique qu'au moment de programmer, il s'intéresse principalement à l'obtention des résultats qui l'aideront dans son projet de recherche ; développer du code qui adhère aux règles strictes d'un logiciel libre comme *Sage* est pour lui d'une importance secondaire.

4.2.3 Le développement d'outils

Dans tous les projets que nous avons décrits jusqu'à présent, les chercheurs choisissent de recourir à la programmation pour s'outiller face à un problème spécifique, appliqué ou pur. Ceci étant dit, il existe certains projets où le développement d'outils informatiques constitue le but principal et non un simple moyen pour une fin. Ces projets ont gagné en importance en raison de l'évolution constante des ordinateurs et de la possibilité de résoudre des problèmes de plus en plus complexes. Nos entrevues nous permettent d'identifier deux types de projets de développement d'outils : (1) le développement d'algorithmes et (2) le développement de logiciels complets.

Dans un projet de type (1), les chercheurs sont impliqués dans la réalisation d'un sous-ensemble des tâches génériques suivantes :

a) développement de l'algorithme ;

b) validation de l'algorithme ;

c) exploration de l'algorithme ;

d) communication des résultats.

Parmi nos participants, nous trouvons deux sous-groupes de mathématiciens. On retrouve d'une part ceux qui apportent une expertise spécifique à une étape particulière. Nathan, par exemple, décrit un projet dans lequel il est un des chercheurs dans un groupe qui propose une nouvelle façon de discrétiser un certain modèle. Bien qu'il contribue à chaque partie du projet, son implication est la plus grande au moment de démontrer la convergence de la méthode (*la validation de l'algorithme (b)*) ; il n'écrit aucune ligne de code. D'autre part, quelques mathématiciens que nous avons rencontrés sont des spécialistes du développement et de l'analyse d'algorithmes. Les sous-tâches clés de la programmation sont toujours *le développement (a) et la validation (b) de l'algorithme*. Mais ces experts écrivent aussi du code pour *explorer l'algorithme (c)* et générer ainsi des exemples de son comportement. Alice explique : « If I'm trying to come up with a new algorithm, then I would prototype it for sure on a computer just to make sure it works and to see how it performs on test cases. But a lot of it is just math, right? » Tout d'abord, la mathématicienne veut fournir à la communauté mathématique un algorithme qui soit théoriquement solide. Au moment de *la communication de ses résultats (d)*, elle explique l'algorithme, présente ses propriétés validées et illustre ces propriétés avec des exemples. Puisque l'algorithme est décrit, les mathématiciens ne ressentent pas nécessairement le besoin de partager le code. Alice suggère cependant que « It's quite tedious to write your own code from scratch; so if somebody is reporting on an algorithm they've developed, it's useful if they give you the code for it. »

Un mathématicien passe au second type de projet lorsqu'il perçoit le besoin ou l'opportunité de combiner en un même répertoire plusieurs algorithmes puissants. Pour ce faire, il accomplit des tâches associées aux types ci-dessous :

a) choix (ou développement) d'algorithmes ;

b) construction du logiciel ;

c) partage du logiciel ;

d) entretien et mise à jour du logiciel.

En premier lieu, il y a *le choix des algorithmes (a)* différents qui formeront la base du logiciel et auxquels on pourrait ajouter de nouveaux algorithmes (type de projet (1)) selon les besoins perçus. Puis commence *la construction du logiciel (b)*, qui est d'un autre niveau dans l'activité de programmation : on détermine la structure des algorithmes (développement d'une architecture), on code dans un certain langage tous les algorithmes et les liens entre eux (codage) et on s'assure enfin que tout fonctionne bien (vérification et validation).

Une fois encore, la participation du chercheur peut varier selon les tâches. Anthony décrit plusieurs moments où il agit comme conseiller auprès de personnes impliquées dans le développement de logiciels de géométrie dynamique ou de conception assistée par ordinateur. Il les aide dans leur *choix (ou développement) d'algorithmes (a)*, en contribuant par ses connaissances théoriques ; tout comme Nathan, il ne code jamais. Alain et Albert, par contre, ont participé à chacune des tâches du processus de développement. Pour Alain, le logiciel est personnel : au fil des ans, il a créé son propre environnement d'éléments finis. Pendant l'entrevue, il a parlé de la tâche énorme *d'entretenir et de mettre à jour ce logiciel (d)*, nécessaire en raison de l'évolution de sa recherche et de la venue de nouveaux algorithmes. Il a récemment éliminé des choses inutiles, changé la structure des données et rendu plus précis certains algorithmes. Même si ces changements requièrent un effort important, ils permettent à Alain de personnaliser et d'ajuster son outil au travail qu'il fait. Albert, qui *partage son logiciel (c)* avec de nombreux mathématiciens dans le monde, doit considérer aussi leurs besoins dans l'entretien de son logiciel. En fait, son logiciel est si important dans son domaine de recherche que tout un groupe de chercheurs participe à son entretien et à sa documentation.

Il convient de noter qu'une étape d'*entretien (d)* peut s'ajouter également aux tâches des chercheurs ayant une bibliothèque personnelle de programmes. Ben, par exemple, a programmé dans plusieurs langages ; s'il ne considère pas avoir eu de difficultés à s'adapter au caractère mouvant de l'informatique en passant d'un langage à l'autre, il comprend la difficulté qui pourrait être éprouvée par ses collègues :

> The *Fortran* programmers never want to program in anything other than *Fortran*. And they'll have a bunch of proprietary libraries that are written in *Fortran.* [...] they know that system perfectly. [...] they can't switch out of it. And they want to teach their students that system. Even though that language is decades and decades old. [...] They're just too invested. [...] These people have spent their whole lives doing research, and good research, using these particular tools. If they were to change, there'd be an immense disruption of what they're working on.

Nous voyons ainsi une autre différence importante entre l'activité mathématique et l'activité de programmation : si la première cherche à développer des produits qui seront éternellement valides, la seconde ne peut espérer (du moins pour le moment) produire des programmes permanents. Ben suggère que certains mathématiciens (p.e., les programmeurs de *Fortran*) essaient quand même d'atteindre une stabilité dans leur environnement de travail en continuant à programmer dans le même langage ; ceci pourrait ralentir par conséquent l'adaptation des pratiques d'enseignement à l'évolution des langages et d'outils informatiques.

4.3 Les pratiques en apprentissage impliquant la programmation

Contrairement à la recherche, l'apprentissage des mathématiques se fait typiquement par la réalisation d'activités ponctuelles plutôt que par la poursuite de longs projets. Par conséquent, au lieu de parler des « processus » comme dans les sections précédentes, nous préférons nous référer à quatre *enjeux d'une formation de premier cycle en mathématiques* pouvant impliquer un recours à la programmation :

E1. appropriation de théories mathématiques abstraites ;

E2. développement de compétences de modélisation et de simulation ;

E3. appropriation de méthodes de calcul et de représentation ;

E4. développement de compétences de programmation.

Nous ne prétendons pas que cette liste soit représentative de tous les enjeux de l'apprentissage des mathématiques au niveau du baccalauréat ; ce sont simplement les enjeux qui, selon nos participants, se révèlent propices à l'utilisation de programmes informatiques. De plus, ces enjeux ne sont pas nécessairement distincts. Par exemple, l'atteinte de E1, E2 ou E3 pourrait exiger le développement de compétences de programmation (E4). En effet, dans une formation mathématique, on n'accorde pas nécessairement la même valeur à chacun de ces enjeux. Personne ne remettrait en question l'importance du premier, qui est souvent vu comme

le but principal d'un baccalauréat en mathématiques. Mais qu'en est-il des autres ? L'ordre des enjeux présenté ci-avant nous semble correspondre à leur poids relatif au sein des programmes de mathématiques. En effet, on peut se demander si le développement de compétences de programmation est vraiment un objectif en soi. Comme nous le verrons plus tard, cette question est cruciale pour mieux comprendre la place de la programmation dans les pratiques en apprentissage.

Néanmoins, la distinction entre ces quatre enjeux se révèle utile pour notre analyse. Remarquons, par exemple, le parallèle entre les enjeux et les pratiques en recherche que nous venons de décrire : la résolution des problèmes purs est associé principalement à E1, la résolution des problèmes appliqués à E2 et les deux derniers enjeux sont liés à tout projet de recherche impliquant la programmation, mais particulièrement ceux qui s'orientent vers le développement d'outils.

L'atteinte de chacun de ces enjeux repose sur l'exposition de l'étudiant à différentes tâches. Dans les exemples décrits par nos participants, nous identifions plusieurs *types de tâches* qui pourraient incorporer la programmation :

T1. réception de résultats mathématiques et de leurs preuves ;
T2. exploration d'objets mathématiques et formulation de conjectures ;
T3. validation d'une propriété ou démonstration d'un théorème ;
T4. traitement de données issues de situations réelles ;
T5. recherche des solutions et expérimentation avec un modèle mathématique ;
T6. analyse ou exploration d'algorithmes ;
T7. réalisation de calculs ou création d'images ;
T8. utilisation, lecture ou écriture de programmes informatiques ;
T9. rédaction de devoirs et passation d'examens.

Une fois encore, il faut souligner que cette liste n'est ni exhaustive, ni composée de tâches distinctes d'importance égale. Par exemple, si certains étudiants de premier cycle sont invités à créer leurs propres modèles mathématiques, tous les exemples décrits par nos participants concernent des modèles existants. De plus, tout comme E4, T8 pourrait être considéré comme un soutien aux autres tâches « plus mathématiques » plutôt qu'une tâche importante en soi.

On peut voir des liens entre ces tâches d'apprentissage et les tâches impliquant la programmation qui sont réalisées par les mathématiciens dans leur recherche. Par exemple, on peut associer T9 à l'étape de communication qui fait partie de n'importe quel projet. On peut trouver également des activités analogues aux tâches T7 et T8 dans tout type de projet. Et des activités plus spécifiques aux différents secteurs de recherche sont représentées principalement par les relations suivantes :

{T2, T3} ↔ la résolution des problèmes purs ;

{T4, T5} ↔ la résolution des problèmes appliqués ;

{T6} ↔ le développement d'outils.

Notons que nos participants mentionnent quelques moments associables à la tâche de recevoir des résultats (T1) : tout comme les étudiants assistent aux cours et font des lectures dans des livres de référence, les chercheurs vont à des conférences chaque année et lisent des articles ou rapports de recherche où ils apprennent des travaux de leurs collègues. Si cet apprentissage des résultats produits par d'autres fait certainement partie importante de ce qu'ils font, les mathématiciens que nous avons rencontrés ne le relèvent pas comme un aspect fondamental pour décrire leur propre recherche. Inversement, T1 est souvent utilisé pour définir le travail d'un étudiant en mathématiques. Il n'est donc pas surprenant que plusieurs mathématiciens comparent leur enseignement à l'aspect de communication de leur travail : dans les deux cas, on cherche à favoriser la compréhension de concepts ou de résultats mathématiques.

Le fait que le travail de l'étudiant soit *orchestré* par le professeur (Trouche, 2004) rend plus complexe la façon dont la programmation est incorporée dans les pratiques en apprentissage. Qui écrit les programmes ? Quel degré d'autonomie est attendu de l'étudiant ? Évidemment, l'accomplissement des tâches énumérées ci-avant (et par extension, l'atteinte des enjeux) n'exige pas nécessairement l'utilisation de programmes informatiques. Lorsque nos participants décident d'en intégrer dans leurs cours, *l'interaction de l'étudiant avec l'activité de programmation* peut se situer à différents niveaux :

N0. observation exclusive des résultats d'un programme informatique ;
N1. manipulation au niveau de l'interface d'un programme informatique ;
N2. observation du code d'un programme informatique ;

N3. modification du code d'un programme informatique ;

N4. conception et écriture du code, dont plusieurs éléments sont spécifiés ;

N5. prise en charge du développement d'un programme informatique.

Plus le niveau est élevé, plus la visibilité de l'activité de programmation est grande et plus l'étudiant y participe. Au niveau N0, toute l'activité de programmation et les programmes qui en résultent sont cachés ; l'étudiant n'interagit pas avec l'ordinateur. Au niveau N1, les programmes deviennent visibles, mais l'activité de programmation reste presque cachée puisque l'étudiant n'a pas accès au code-source ; il demeure au niveau de l'interface-usager. La programmation devient accessible à l'étudiant au niveau N2, mais l'activité reste encore partiellement cachée : l'étudiant sait qu'il y a eu une activité de programmation parce qu'il voit le code et il peut même réussir à comprendre des éléments de ce code et de l'interpréteur utilisé, mais il n'en saisit pas toutes les subtilités. Finalement, au niveau N3, l'étudiant devient actif dans l'activité de programmation : il joue avec le code et en change des éléments pour accomplir de nouvelles tâches. Une connaissance du langage de programmation utilisé devient ainsi importante. Aux niveaux N4 et N5, l'étudiant prend en charge une part importante ou même l'ensemble de l'activité de programmation, avec une autonomie croissante dans le processus de décision et de conception. Dans ces cas, l'étudiant pourrait avoir à accéder aux connaissances de tous les niveaux des programmes informatiques énumérés en Chapitre 2 (voir la page 11), y compris ceux du système d'exploitation et des composantes physiques de l'ordinateur. En effet, notons qu'il y a une correspondance entre les niveaux d'interaction avec la programmation et les niveaux différents d'un programme informatique. Tout type d'interaction de l'étudiant avec la programmation pourrait donc contribuer au développement de ses compétences de programmation et de ses schèmes d'instrumentalisation de l'ordinateur. Pourtant, comme plusieurs de nos participants le suggèrent, et comme le confirment des travaux de recherche (Artigue, 2002), la programmation ne peut devenir naturelle pour l'étudiant qu'après une exposition prolongée aux niveaux supérieurs (N3, N4 et, éventuellement, N5).

Les listes que nous venons de construire nous servent, dans les sections suivantes, à décrire comment les professeurs de notre étude intègrent la programmation dans les pratiques qu'ils planifient pour leurs étudiants. Pour chacun des enjeux (E) visés par leur enseignement,

nous examinons les types de tâches proposées (T) et les niveaux (N) auxquels les étudiants sont invités à interagir avec l'activité de programmation.

4.3.1 L'appropriation de la théorie mathématique abstraite

De la réception de résultats (T1) à une exploration guidée (T2)

Comme nous l'avons déjà mentionné, recevoir des résultats mathématiques et leurs preuves est une tâche très fréquente dans la vie d'un étudiant ; en fait, l'apprentissage en classe de mathématiques est parfois réduit à la tâche de « recevoir » beaucoup d'informations sur des savoirs souvent constitués il y a des centaines d'années. Mais le mot « réception » ne signifie pas pour autant que l'étudiant soit toujours passif pendant ces cours-conférences. L'accomplissement de cette tâche peut solliciter, à des degrés divers, une activité de la part de l'étudiant. Dans certains cas, le professeur communique les savoirs directement aux étudiants, qui les reçoivent généralement de façon passive. Nous avons tous vécu de telles séances dans lesquelles se succèdent au tableau définitions, théorèmes et preuves. Dans d'autres cas, même à l'intérieur d'un cours magistral, le professeur, par son approche didactique, sollicite davantage l'activité des étudiants, au point où ceux-ci ont l'impression d'explorer un problème mathématique. Ils sont si bien guidés par le mathématicien qu'ils arrivent à formuler rapidement des conjectures et à s'engager dans leur validation. Ils vivent des moments de confusion, de surprise, de questionnement, d'illumination, comme en vivent les mathématiciens dans leur propre recherche. Après tout, même si les résultats sont déjà bien connus du professeur et bien documentés dans la littérature, il est fort probable qu'ils ne sont pas connus des étudiants au moment de leur arrivée en classe.

Pour aider l'étudiant à comprendre les savoirs qu'il cherche à lui transmettre, le professeur peut choisir de recourir à la programmation, typiquement pour illustrer avec des images ou des animations portant sur des exemples spécifiques, le comportement ou les propriétés des objets mathématiques enseignés. Un tel travail de programmation par le professeur peut faire en sorte que la « réception » par les étudiants des résultats mathématiques soit vécue comme une « exploration » guidée, plus propice à la compréhension. L'exemple suivant, décrit par Omar pour son cours d'algèbre linéaire, permet d'en illustrer les possibilités.

Omar base sa présentation sur la définition suivante de l'espace engendré par un ensemble de vecteurs :

> *Définition : l'espace engendré par les vecteurs* $a_1, a_2, \ldots, a_p$ *est l'ensemble de toutes les combinaisons linéaires de ces vecteurs, c'est-à-dire des vecteurs de la forme* $w = k_1 a_1 + k_2 a_2 + \cdots + k_p a_p$*, où* $k_1, k_2, \ldots, k_p$ *sont des scalaires.*

Après avoir donné cette définition, qui peut sembler abstraite et difficile pour plusieurs étudiants, Omar va à son ordinateur, où il fait appel à un programme informatique qu'il a déjà préparé. Les étudiants voient alors apparaître sur l'écran l'image ci-dessous, précédée d'un extrait du code *Mathematica* écrit par leur professeur :

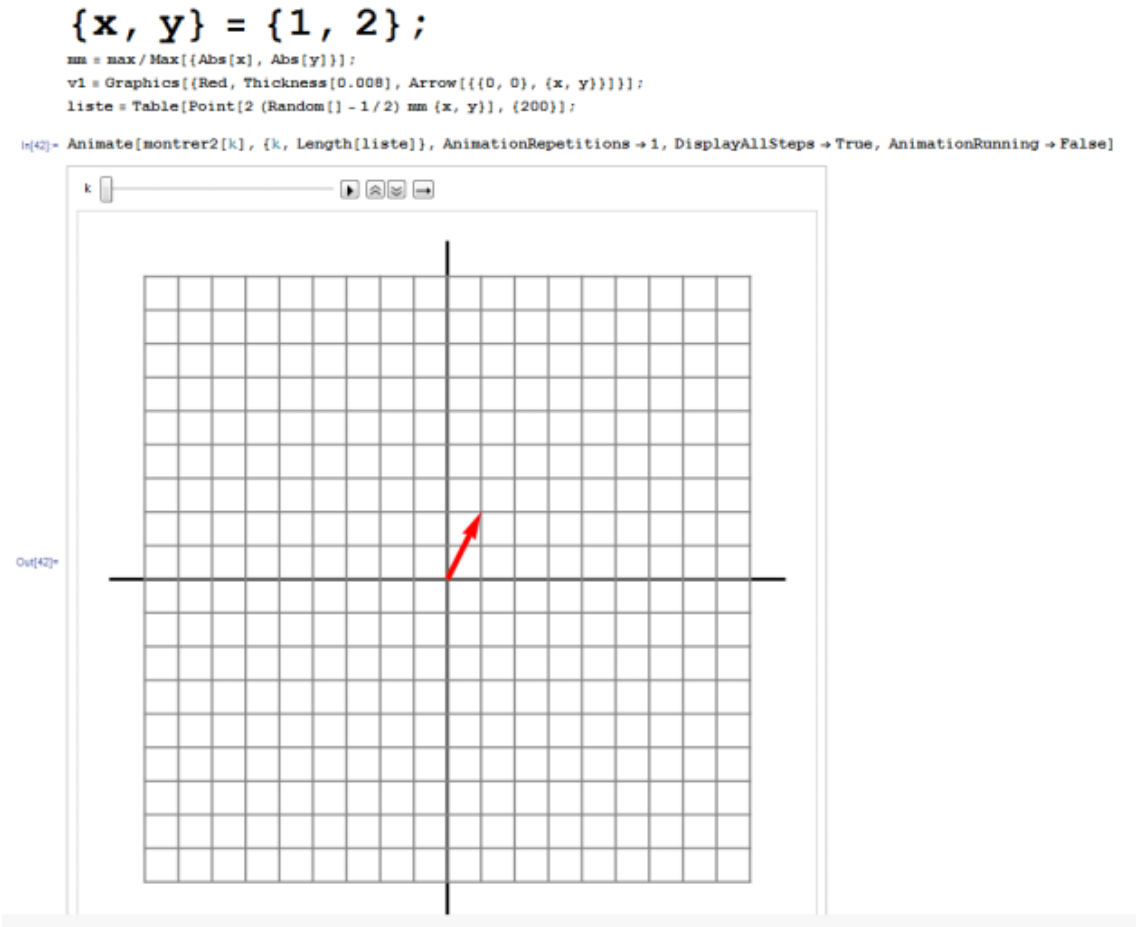


Figure 4.14 L'écran initial observé par les étudiants d'Omar.
Le code est à peine lisible ; le professeur attire plutôt l'attention de ses étudiants vers les vecteurs utilisés.

Omar commence avec un exemple simple : l'espace engendré par un seul vecteur à deux dimensions. Il prépare ses étudiants à ce qu'ils observeront en expliquant qu'il est impossible de faire afficher toutes les combinaisons linéaires du vecteur $(1, 2)$ parce qu'il y en a une infinité. Par conséquent, il fera calculer par son programme un grand nombre de combinaisons linéaires et dessinera sur le graphique les extrémités des différents vecteurs résultants. À ce moment, nous pouvons imaginer que les étudiants (tout au moins ceux qui sont en train d'écouter) formulent des conjectures sur ce qui va se passer. Omar clique sur un bouton pour démarrer une animation qui leur permet de vérifier leurs conjectures :

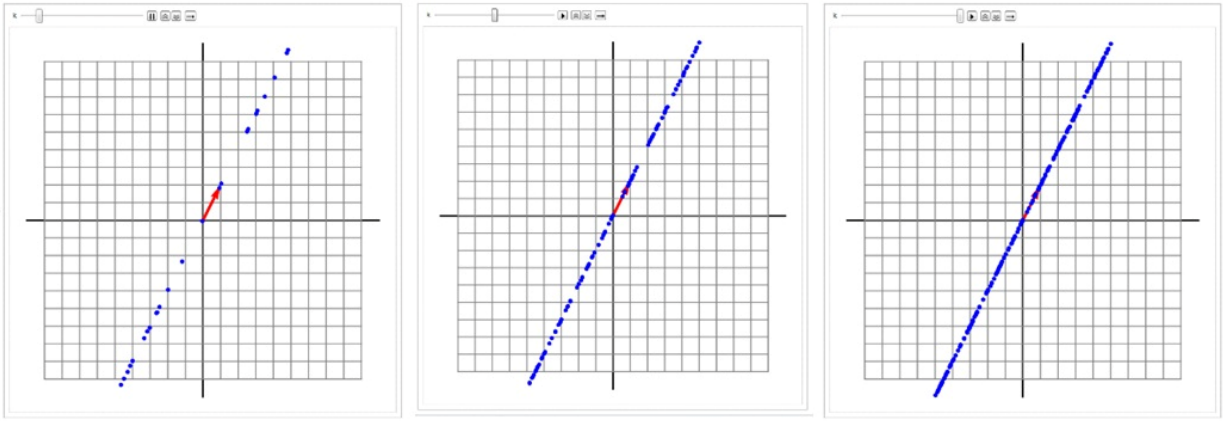


Figure 4.15 L'animation des combinaisons linéaires du vecteur $(1, 2)$.

Omar explique :

> Et là je demande aux étudiants, « Bien, ça va finir comment si je continue ? » Puis ça c'est évident : il va finir la ligne au complet. Donc je dis « Ok, donc, ça a l'air que l'espace vectoriel engendré par un vecteur, c'est la droite qui est le long de ce vecteur-là. » À tout hasard, je suggère que nous en fassions un autre.

Ayant un accès direct à son code pendant cette présentation, Omar peut facilement changer le vecteur et générer une autre animation comme celle illustrée par les images ci-dessus. Il dit que la visualisation de ces deux exemples est d'habitude assez convaincante pour que tout le monde accepte et comprenne que l'espace engendré par un seul vecteur en deux dimensions est une droite.

Selon les réactions des étudiants et le temps qui reste à la séance, Omar détermine les prochaines étapes. Il peut décider de traiter l'espace engendré par deux vecteurs, toujours en deux dimensions, ou de passer immédiatement aux espaces engendrés par des vecteurs en trois

dimensions. Une fois en 3D, il commence encore avec un seul vecteur et il demande immédiatement aux étudiants : « Est-ce que vous pouvez deviner ce que ça va engendrer quand je vais prendre toutes les combinaisons linéaires d'un vecteur ? » Forts des observations qu'ils viennent de faire, les étudiants répondent typiquement avec la bonne réponse : « Bon, ça va donner une droite. » Ils ne sont donc pas surpris de voir l'image ci-dessous générée par leur professeur :

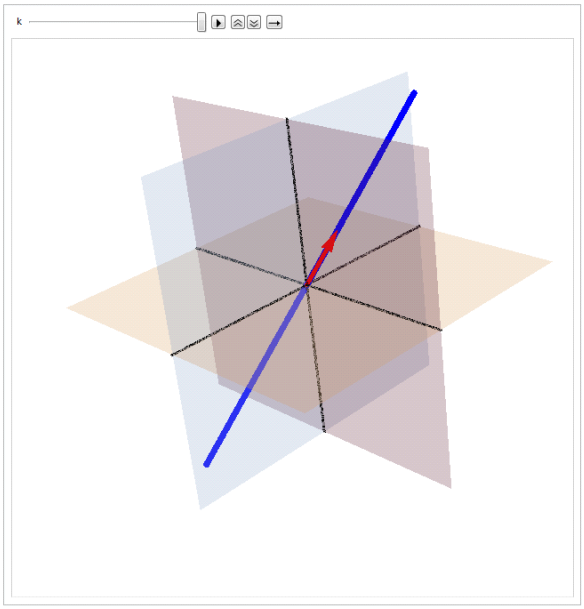


Figure 4.16 L'espace engendré par un vecteur en trois dimensions.

Omar explique qu'à ce moment, la plupart des étudiants sont en mesure de formuler les conjectures selon lesquelles l'espace engendré par deux vecteurs est un plan et l'espace engendré par trois vecteurs est tout l'espace. Un exemple avec deux vecteurs sert à accroître leur confiance envers leurs conjectures. Omar ajoute un second vecteur au diagramme à la Figure 4.16, encourage un peu d'interaction et fait l'animation correspondante. À cause des capacités graphiques de *Mathematica*, il est capable d'appliquer une rotation au repère de l'image finale pour avoir la meilleure perspective possible et donc de soutenir la compréhension de ses étudiants (Figure 4.17). Si les exemples en une dimension peuvent être aisément présentés à l'aide d'une simple craie et d'un tableau, l'exemple ci-après montre clairement le potentiel de l'ordinateur pour générer des représentations graphiques plus précises, plus claires et facilement manipulables.

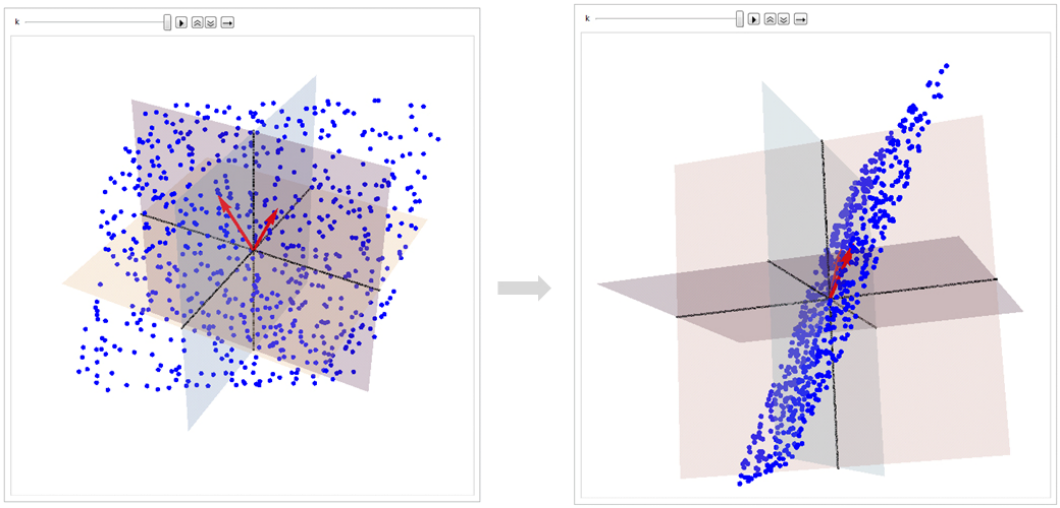


Figure 4.17 La recherche d'une bonne perspective de l'espace engendré par deux vecteurs en trois dimensions.

Après cet exemple, Omar ajoute encore un troisième vecteur au programme :

```
max = 10;
pmax = 1.01 max;
grille = Graphics3D[{GrayLevel[0.8], Opacity[0.3], EdgeForm[{}], Polygon[{{-max, -max, 0}, {max, -max, 0}, {max, max, 0}, {-max, max, 0}}],
    Polygon[{{-max, 0, -max}, {max, 0, -max}, {max, 0, max}, {-max, 0, max}}],
    Polygon[{{0, -max, -max}, {0, max, -max}, {0, max, max}, {0, -max, max}}],
    Black, Thickness[0.005], Line[{{-max-1, 0, 0}, {max+1, 0, 0}}],
    Line[{{0, -max-1, 0}, {0, max+1, 0}}], Line[{{0, 0, -max-1}, {0, 0, max+1}}]
   }];
v1 = {1, 2, 3};
v2 = {-3, 0, 4};
v3 = {-2, 2, 7};
nv1 = N[v1/Norm[v1]];
nv2 = N[v2/Norm[v2]];
nv3 = N[v3/Norm[v3]];
mM = max*2.7;
liste3D = Table[Point[mM ((Random[]-1/2) nv1 + (Random[]-1/2) nv2 + (Random[]-1/2) nv3)], {1200}];
lesPoints = Graphics3D[{Blue, PointSize[0.012], liste3D}];
vecteurs = Graphics3D[{Red, Thickness[0.008], Arrow[{{0, 0, 0}, v1}], Arrow[{{0, 0, 0}, v2}], Arrow[{{0, 0, 0}, v3}]}];
(* Show[grille,vecteurs,lesPoints,
 Boxed→False,PlotRange→{{-pmax,pmax},{-pmax,pmax},{-pmax,pmax}}] *)
Animate[montrer3[k], {k, Length[liste3D]}, AnimationRepetitions→1, DisplayAllSteps→True, AnimationRunning→False]
```

Figure 4.18 L'ajout d'un troisième vecteur au fichier *Mathematica*.

Comme d'habitude, avant de démarrer l'animation, il encourage la formulation de conjectures par les étudiants. Il explique : « Je passe au vote : "Qu'est-ce que ça va être ? Ça va être un plan ? Une droite ?" Puis là tout le monde me dit "Ah, non, non, non. C'est clair que ça va être tout !" » Les étudiants sont pris au piège ; comme v1 + v2 = v3, le troisième vecteur n'ajoute rien à l'espace engendré par les deux premiers, lequel demeure un plan (voir la Figure 4.19).

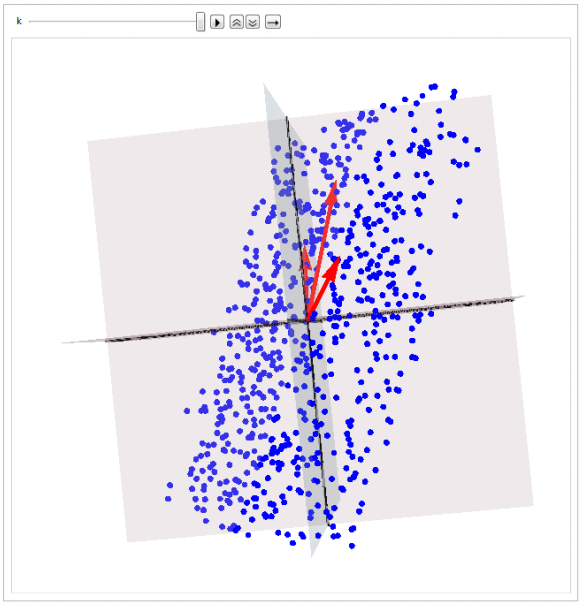


Figure 4.19 Un plan en trois dimensions engendré par trois vecteurs.

À ce stade du cours, le concept d'indépendance linéaire, qui aurait pu aider les étudiants à deviner le bon résultat, n'a pas encore été abordé. Mais « deviner le bon résultat » n'est pas le but de l'exercice ; Omar cherche à déséquilibrer ses étudiants. Comme stratégie didactique, cette confrontation à un exemple surprenant poursuit deux buts :

1) poser les bases d'une discussion informelle autour d'une idée plus abstraite (l'indépendance linéaire dans ce cas) qu'on formalisera plus tard dans le cours ;

2) encourager les étudiants à reformuler leurs conjectures.

Omar explique qu'il y a toujours quelques étudiants qui s'exclament : « Tu ne pourras jamais tout remplir ! » Omar se fait donc un plaisir de répondre à ce défi en montrant un exemple final de trois vecteurs (linéairement indépendants) qui engendrent tout l'espace, ce dont les étudiants peuvent être convaincus par de nouvelles rotations du repère (voir la Figure 4.20 ci-après).

Enfin, en ayant consacré à peine dix minutes à cette exploration, Omar a le sentiment d'avoir pu expliquer de façon très efficace le concept d'espace engendré par un ensemble de vecteurs ; il voit dans les interactions avec les étudiants qu'ils tendent à comprendre.

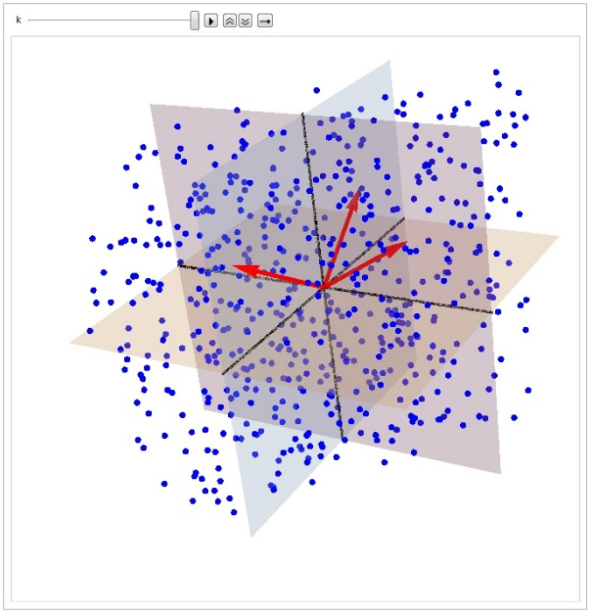


Figure 4.20 Trois vecteurs qui engendrent tout l'espace en trois dimensions.

Il affirme de plus que les étudiants qui n'ont jamais appris ou compris ce concept « réalisent quelque chose de crucial qui va rester longtemps dans leur cerveau. » L'utilisation de son programme informatique lui permet d'enseigner le concept de façon marquante, mais aussi beaucoup plus intéressante qu'avec une simple craie et un tableau. Il en témoigne avec l'analogie suivante :

> C'est comme apprendre comment changer les bougies d'allumage à l'intérieur d'une voiture au tableau. Tout le monde dit « Bon. Trop abstrait. » Là tu ouvres l'auto, puis tu dis aux étudiants « Regardez tous là, c'est ici. » Puis là les gens vont dire « Ah, tasse-toi, je veux le voir ! » Mais, quand tu es au tableau, c'est : ok, on copie, c'est abstrait...

En somme, la justification que donne Omar à l'intégration de la programmation dans son enseignement est essentiellement *épistémique*. En permettant aux étudiants de visualiser à travers plusieurs exemples un concept passablement abstrait et non trivial, il cherche à favoriser leur appropriation de la théorie enseignée. Il attribue même une valeur épistémique supérieure à l'animation de ces images et des objets auxquels elles renvoient :

> Seeing things in motion is extremely useful to be able read things properly, rather than having five pictures of the same object at different points of view. That doesn't do the same thing as actually dragging the object in front of the students.

S'il voit aussi un apport *pragmatique* de la programmation, il le situe du côté de l'enseignant : de telles images seraient difficiles à produire sans ordinateur et il n'y aurait pas une telle facilité pour en changer les paramètres, les animer en classe et les utiliser de façon dynamique pour mieux répondre aux questions qui peuvent surgir de la classe.

D'autres professeurs parmi ceux que nous avons rencontrés utilisent la programmation d'une façon semblable à celle d'Omar, pour rendre leurs séances plus intéressantes et plus marquantes, pour aider les étudiants à comprendre des idées abstraites, souvent à l'aide d'exemples dynamiques, et pour motiver l'apprentissage de concepts théoriques à venir. Tout comme Omar, plusieurs mathématiciens valorisent aussi l'élément de surprise, la rencontre du fait surprenant : la nécessité de défier « l'intuition » des étudiants et de leur montrer ultimement qu'on ne peut pas utiliser des résultats mathématiques aveuglément. Par exemple, Pierre intègre une exploration très voisine à celle d'Omar dans son cours de calcul, non seulement pour montrer la signification du théorème de Taylor, mais aussi pour faire ressortir les limites et les conditions d'application de cette idée. Rappelons-nous le théorème de Taylor qui est typiquement vu dans la première année d'un baccalauréat en mathématiques :

Théorème : Soient $k \geq 1$ *un entier et* $f : R \to R$ *une fonction qui est différentiable* k *fois au point* $a \in R$. *Alors, il existe une fonction* $h_k : R \to R$ *telle que*

$$f(x) = \sum_{j=0}^{k-1} \frac{f^{(j)}(a)}{j!}(x-a)^j + h_k(x)(x-a)^k$$

et

$$lim_{x \to a}\, h_k(x) = 0\,.$$

La fonction

$$T_n(x) = \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j$$

est appelée le n*-ième polynôme de Taylor pour* $f(x)$ *autour de* a.

À l'aide d'un exemple simple, Pierre est capable de rendre concret l'énoncé de ce théorème qui peut sembler très abstrait pour des étudiants de première année. Avec un programme qu'il a déjà conçu en *Sage*, il montre aux étudiants la représentation graphique de la fonction $f(x) = \sin(x)$ et son polynôme de Taylor de degré $n = 4$ (par exemple) autour de l'origine ($a = 0$). Sur le graphique, les étudiants peuvent voir que le polynôme de Taylor constitue une très bonne approximation de $f(x)$ près de l'origine (voir le premier graphique à la Figure 4.21). Dans son programme, Pierre a aussi créé un curseur qui lui permet d'augmenter la valeur de *n* et de montrer aux étudiants les graphiques résultants, l'un après l'autre et de façon animée. La courbe du polynôme semble alors épouser la fonction f, ce qui convainc les étudiants que l'approximation d'une fonction par son polynôme de Taylor gagne en précision avec l'augmentation du degré du polynôme.

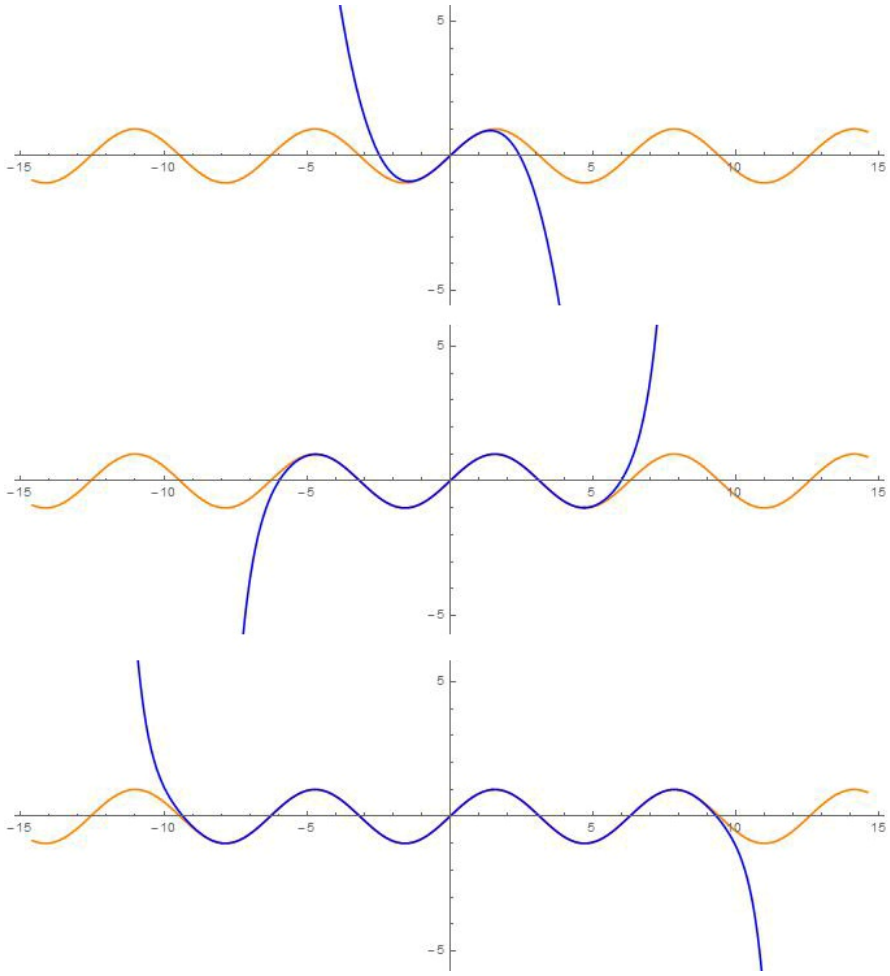


Figure 4.21 Un exemple de la puissance du théorème de Taylor : $f(x) = \sin(x)$ en orange, $T_n(x)$ en bleu pour $n = 4, 14, 24$, de haut en bas.

Malheureusement, ce n'est pas le cas pour toutes les fonctions possibles et les étudiants n'en sont pas encore conscients. Un exemple de plus les invite à revoir les conjectures fausses qu'ils ont possiblement faites. Pierre change la fonction directement dans son code-source pour $f(x) = \arctan(x)$ et est très prudent de n'afficher au départ qu'un petit intervalle sur l'axe des x. De cette façon les étudiants sont typiquement amenés à croire qu'il n'y a rien de différent dans cet exemple : le polynôme de Taylor devient une meilleure approximation à la fonction au fur et à mesure que la valeur de n devient plus large. Mais en étendant le domaine affiché, les étudiants voient quelque chose qui pourrait leur sembler bizarre à première vue : le polynôme de Taylor autour de l'origine ne devient une meilleure approximation à la fonction que sur un intervalle très restreint (voir la Figure 4.22 ci-dessous).

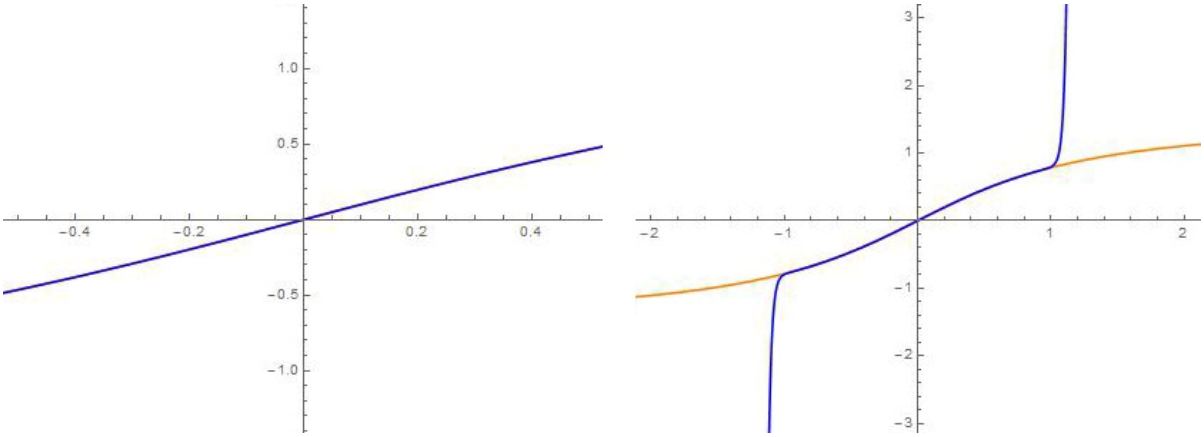
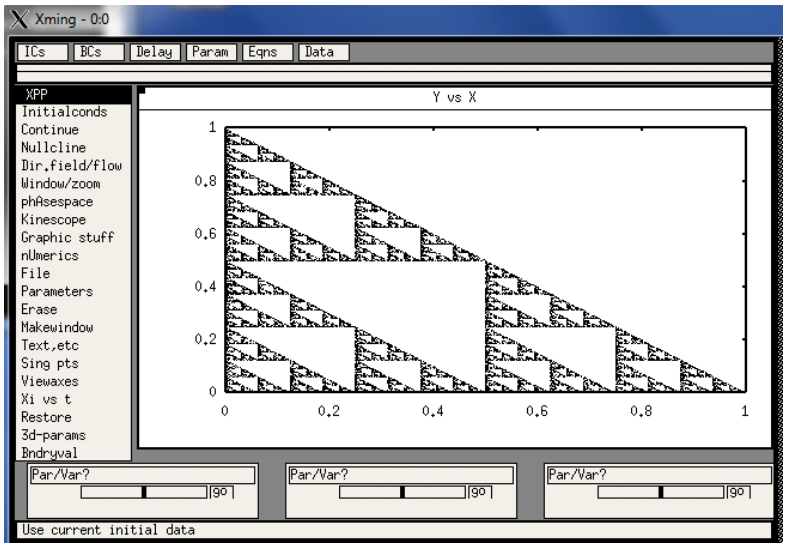


Figure 4.22 Un exemple pour montrer les limites du théorème de Taylor : $f(x) = \arctan(x)$ en orange, $T_n(x)$ en bleu pour $n = 50$.

Cet exemple donne à Pierre l'occasion d'établir des liens avec d'autres concepts théoriques déjà vus en classe (le rayon de convergence, notamment), d'en souligner l'importance et, enfin, de montrer aux étudiants que tout résultat mathématique a ses limites. Albert et Philippe utilisent des programmes informatiques qu'ils ont développés pour faire presque la même chose que Pierre, mais autour de grandes idées en probabilités : par exemple, le théorème limite central et la loi des grands nombres. Philippe dit de son approche de montrer des exemples contre-intuitifs aux étudiants : « Je veux qu'ils apprennent à penser à ce qu'ils font. […] pas nécessairement se fier à l'intuition, mais se fier à ce qu'on pense ou à ce qu'on a étudié. » Ses exemples poussent les étudiants à toujours vérifier les hypothèses d'un théorème avant de l'appliquer.




```
#sierpinski.ode
#drawing the Sierpinski triangle
#
par c0=0,c1=.5,c2=0
par d0=0,d1=0,d2=.5
p=flr(ran(1)*3)
dx/dt=.5*x+shift(c0,p)
dy/dt=.5*y+shift(d0,p)
@ xp=cx,yp=cy
@ maxstor=20000000,total=20000,
meth=discrete
@ xlo=0,xhi=1,ylo=0,yhi=1,lt=0
@ xp=x,yp=y
aux pp=p
done
```

Figure 4.23 Le triangle de Sierpinski et le code en *XPPAUT* qui le génère.

Notons qu'il arrive souvent que « l'exploration » faite avec la programmation pour aider les étudiants à comprendre des idées abstraites soit beaucoup plus limitée que dans les exemples précédents. Plusieurs participants mentionnent que la génération d'une seule image saisissante peut suffire à engager une certaine discussion. Barbara, par exemple, motive une séance complète sur les fractales en exécutant un programme qui génère le triangle de Sierpinski (Figure 4.23).

Dans toutes ces approches d'enseignement où le professeur crée des programmes à des fins *épistémiques*, pour présenter de nouvelles notions, le niveau d'interaction de l'étudiant avec la programmation reste typiquement au niveau N0 : l'étudiant observe les résultats d'un programme informatique, avec ou sans la manipulation au niveau de l'interface par le professeur. Même dans les exemples d'Omar et de Pierre, où le code des programmes est parfois visible pendant leurs présentations, ce code reste inaccessible à l'étudiant ; cela s'explique sans doute par le fait que le but visé (l'aide à la compréhension d'un concept mathématique) a déjà été atteint avec la courte utilisation du code en classe. On peut y voir un parallèle avec les pratiques en recherche où l'on ne partage pas toujours les programmes utilisés pour développer des théories abstraites. Albert, dont les programmes utilisés dans ses cours ne sont montrés qu'au niveau de l'interface, explique qu'il ne les partage pas avec ses étudiants car il n'a pas le temps de les rendre efficaces ou d'y ajouter la documentation nécessaire. Rappelons que ceci est un obstacle mentionné par plusieurs chercheurs au regard

du partage des programmes dans leur travail de recherche. Albert est pourtant très critique de ses pratiques d'enseignement et reconnaît lui-même la contradiction entre son choix de ne pas partager ses programmes avec ses étudiants et l'effort considérable qu'il fait pour partager avec la communauté mathématique les programmes qu'il développe dans sa recherche. Il conclut donc qu'il serait important de rendre visible le code qu'il utilise en classe, pas seulement pour aider les étudiants à mieux comprendre ce qu'il fait, mais aussi pour leur donner une ressource utile. Il dit : « I think it would help some students. Some might be motivated to modify [the programs] or to redo them or to make them more general or something. » Ainsi, Albert voit le partage de son code comme une possibilité d'encourager les étudiants à interagir à un niveau plus haut (N3) avec la programmation.

Certains mathématiciens nous ont confié avoir provoqué une progression dans les niveaux d'interaction. Un exemple intéressant est celui de Nathan, qui a décidé récemment de passer d'une présentation exclusive de résultats (N0) à une plus grande visibilité de l'activité de programmation (N1 et N2). Ce mathématicien construit lui-même de petits programmes en *R* pour expliquer aux étudiants des idées plus abstraites dans son cours de mathématiques financières. Il a toujours employé des images pour aider les étudiants à visualiser ces concepts et, jusqu'à récemment, il arrivait en classe avec des images déjà créées qu'il projetait sur un écran, cachant complètement son activité de programmation. Mais maintenant il commence à apporter ses codes informatiques, à les montrer et à les exécuter en classe pour générer les images devant ses étudiants. À titre d'exemple, la Figure 4.24 ci-après montre le programme qu'utilise Nathan pour produire plusieurs graphiques qui lui servent à expliquer le concept de mouvement Brownien. Mais Nathan ne montre pas seulement ce code en classe : il le rend disponible après et invite ses étudiants à l'utiliser. Il leur dit : « Jouez à la maison, faites rouler le code souvent, faites varier les paramètres, regardez c'est quoi l'impact de tel paramètre sur la trajectoire qui sera générée, essayez de gagner une intuition de ça. » Contrairement à Albert, Nathan ne s'inquiète pas de l'efficacité de ses programmes. En fait, il suggère que c'est parfois mieux si les programmes restent naïfs pour que les étudiants en comprennent bien les étapes. Évidemment, ceci n'est possible qu'en raison de la simplicité des programmes employés. L'utilisation de programmes plus complexes, comme ceux écrits par des mathématiciens dans leur recherche, exigerait une optimisation du code, moins accessible au néophyte.

```
#
#simulation d'un mouvement brownien
standard
#set.seed(1)
T=1
n=1000 #nb periodes
dt=T/n #pas de temps
Xzero=0 #valeur initiale
dX=rnorm(n,0,sqrt(dt))
X=Xzero+c(0,cumsum(dX))
t=0:n
plot(t/n,X,type="l",col="red",xlab="temps",
ylab="",main="Trajectoire d'un mouvement
brownien standard")
#
```

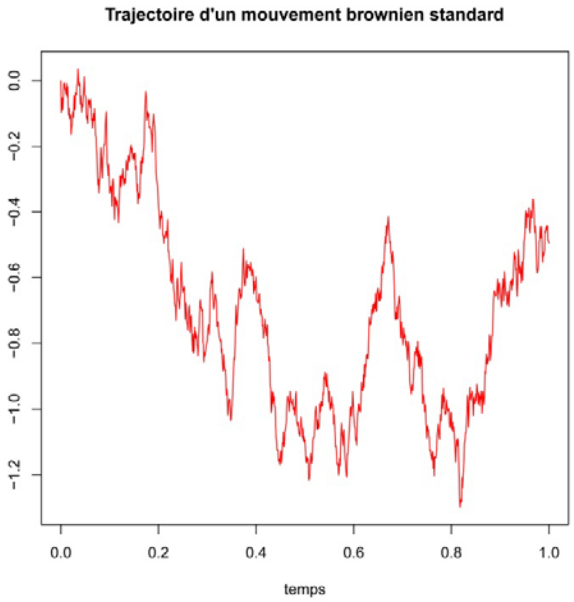


Figure 4.24 Le code en *R* et un exemple du type d'images que Nathan utilise pour expliquer le mouvement Brownien.

Philippe et Barbara essaient aussi d'encourager une interaction plus riche avec la programmation : ils expliquent leurs programmes aux étudiants en classe et les mettent en ligne après. Ces mathématiciens ajoutent qu'en observant le code des programmes (N2), les étudiants peuvent au moins surmonter leur peur de programmer. Barbara explique : « I want them to see that programming isn't that bad. You can do lots of interesting stuff with just a few lines of code. » Philippe partage cette vision :

> J'espère que seulement le fait de montrer que [la programmation] est simple, qu'on peut comprendre plusieurs choses, puis qu'on peut voir ce qui se passe, j'espère que c'est une motivation suffisante pour les encourager à le faire. Mais je ne donne pas de points supplémentaires, je n'ai pas d'incitatifs autres que leur développement.

L'étudiant a ainsi l'occasion d'interagir avec la programmation à un niveau plus haut (p.e., N1 ou même N3), mais seulement s'il en prend l'initiative.

Une question émerge alors : pourquoi ne pas exiger des étudiants qu'ils participent davantage à l'activité de programmation dans les exemples précédents ? Par exemple, on peut se demander si l'exploration faite par Pierre concernant le théorème de Taylor ne pourrait pas être traduite en une activité faite complètement par ses étudiants (N4 ou N5) ; pendant notre projet, nous avons essayé de développer une telle exploration (voir Annexe F). Plus tard dans

l'entrevue, Albert suggère que sa propre illustration du théorème limite central ferait un bon devoir : « This would be a good assignment for the students. They should do that, and say "Wow!" And they will never forget what the central limit theorem is. » Seulement trois mathématiciens de notre étude ont décidé d'intégrer des activités où leurs étudiants doivent programmer dans le but d'explorer des idées mathématiques abstraites.

Exploration autonome (T2)

Pour amener ses étudiants un peu plus loin dans l'exploration du code et des objets mathématiques abstraits auxquels il donne accès, Barbara leur offre plusieurs occasions d'accéder à ses programmes et de jouer avec leurs paramètres (N1) ou des parties du code (N3). Par exemple, quelques devoirs dans son cours de systèmes dynamiques demandent aux étudiants de changer des aspects de son code pour explorer le comportement des systèmes et en chercher des propriétés spécifiques (les points fixes, le comportement chaotique, les doublements de périodes, les points de bifurcation, etc.). Le but, explique-t-elle, est que les étudiants en arrivent à comprendre les différents types de portraits de phase et les dynamiques possibles auxquels ils renvoient. De plus, pour finir son cours, Barbara met sur pied un concours pour couronner les plus belles images des ensembles de Julia à partir d'un programme qu'elle a écrit et dont les étudiants peuvent changer les paramètres. Sans être très exigeant sur le plan de la rigueur, ce travail, qui peut permettre autant de mieux comprendre le rôle de ces paramètres que de récolter des points supplémentaires, connaît un vif succès auprès des étudiants et donne lieu à de magnifiques productions :

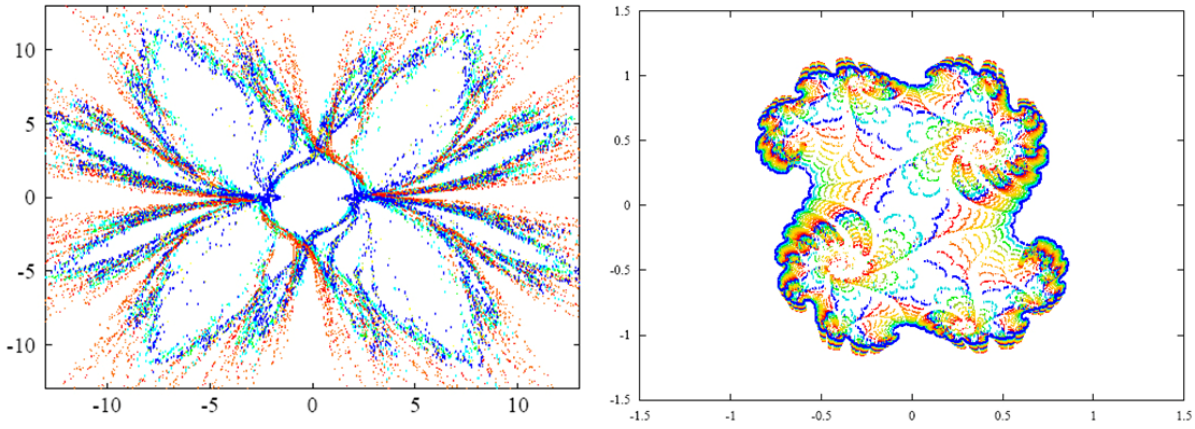


Figure 4.25 Quelques ensembles de Julia créés par les étudiants de Barbara.

Dans la même veine, Omar offre à ses étudiants un concours optionnel pour les encourager à programmer avec *Mathematica* la création d'une image qui puisse aider à expliquer un concept mathématique de leur choix. Si ce concours ne connaît pas un taux de participation aussi fort que celui de Barbara et si son statut demeure extracurriculaire, il est intéressant de noter qu'Omar laisse entièrement aux étudiants le soin de créer leurs propres programmes (N4). Omar explique qu'il peut facilement suggérer la programmation chez ses étudiants puisqu'ils ont tous suivi un cours d'introduction à *Mathematica* ; Barbara, par contre, ne demande pas à ses étudiants de programmer, car ils n'y ont pas tous été initiés.

Paul est le seul mathématicien qui mentionne exiger de ses étudiants d'un cours de premier cycle qu'ils écrivent des programmes eux-mêmes (N4) pour apprendre des éléments d'une théorie abstraite. Cela se fait dans un cours de probabilités, où chacun des devoirs demande de programmer dans un environnement qui s'appelle *Octave.* Cette intégration de la programmation peut être mise directement en comparaison avec celle d'Albert ou celle de Philippe, deux professeurs qui enseignent des cours semblables, mais qui exigent des niveaux plus bas d'interaction de l'étudiant avec l'activité de programmation. De façon intéressante, les premières raisons données par Paul pour impliquer ses étudiants dans l'écriture de programmes sont très proches de celles invoquées par ces autres mathématiciens pour justifier leur propre programmation dans la préparation de leurs cours, et renvoient donc, elles aussi, à une justification essentiellement *épistémique* :

> The purpose was to get the students actively engaged in observing the random phenomena [...] for them to actually see this happen is very meaningful. So it's one thing to explain with equations and so on that this is what's going to happen. [...] But, for the students, it's kind of like, "Well yeah, that's fine, it's a bunch of equations and stuff like that, some sort of argument, I don't totally understand what you just said..." But if they just look at this graph, then they just know what it is.

Tout comme Albert et Philippe, ce professeur adhère ainsi à l'idée selon laquelle les expériences d'exploration à l'aide de l'ordinateur aident à rendre l'apprentissage de l'étudiant plus intéressant, plus marquant et plus efficace. Il faut donc se demander ici s'il est vraiment nécessaire que ce soit les étudiants qui aient programmé eux-mêmes l'outil d'exploration qu'ils utilisent. Paul en est convaincu, mais la plupart des raisons qu'il invoque pourraient à

nouveau s'appliquer tout autant à l'exploration autonome par les étudiants à l'aide d'une simple application, déjà programmée :

> It's much better if the students can program it for themselves. If they're sitting in front of the screen and they can play with it and they can adjust parameters, it becomes a kind of a game and it's more interactive for them. And it's better than me just showing them a picture at the front of the classroom, you know what I mean? If they're doing it themselves, they learn it way better.

Cette idée « d'apprendre beaucoup mieux en programmant » nécessite plus d'élaboration. Comme nous le verrons dans les prochaines sections, les réflexions d'autres participants pourraient ajouter à celles de Paul.

Pour l'instant, nous nous contentons d'utiliser un exemple des questions qu'il pose à ses étudiants pour illustrer leur interaction avec la programmation. Les questions suivantes se trouvent dans un devoir :

1) Simuler 10 000 fois une variable aléatoire normale standard et créer un graphique de la moyenne mobile, c'est-à-dire de $(X_1 + \ldots + X_n)/n$ en fonction de n, où X_i est la i-ième valeur simulée et n est pris entre 1 et 10 000. Le graphique apparaît-il converger vers 0 ?

2) Soit Y une variable distribuée uniformément sur l'intervalle $(-\pi/2, \pi/2)$. La variable $X = \tan Y$ n'a pas d'espérance, même si elle semble avoir une espérance de 0. Pour l'illustrer, simuler X 10 000 fois et faire un graphique de la moyenne mobile. Paraît-elle converger vers 0 ? Paraît-elle converger ?

Comme dans tous les exemples déjà élaborés, nous observons la main « bienveillante » du professeur qui orchestre l'exploration de l'étudiant. Mais cette fois-ci c'est l'étudiant qui doit concevoir le programme et prendre plusieurs décisions à cet effet. Afin de mieux comprendre le type de processus dans lequel un étudiant de Paul pourrait s'engager en complétant ces activités, nous avons décidé de recréer l'exploration nous-mêmes. Évidemment, tous les étudiants n'exploreront pas ces problèmes de la même façon ou au même degré. Il est vrai que les étudiants vont rarement plus loin que ce qui est exigé. Alors, le questionnement d'un étudiant typique pourrait être moindre que ce que nous allons montrer. Néanmoins, l'explication suivante de notre travail souligne le fait qu'en écrivant un code assez

simple, un étudiant pourrait vivre des explorations enrichissantes semblables à celles menées par les mathématiciens lorsqu'ils cherchent à développer de nouvelles théories abstraites.

Pour répondre à la première question, nous écrivons le code ci-dessous, assorti de nos commentaires[5] :

$x = RandomVariate[NormalDistribution[],10000];$
(*simuler une variable aléatoire normale standard 10 000 fois*)
$runningaverage[n\_] := 1 / n * Sum[x[[i]], \{i, 1, n\}];$
(*calculer la moyenne mobile*)
$Plot[runningaverage[n], \{n, 1,10000\}]$
(*faire un graphique de la moyenne mobile*)

En dépit de sa simplicité, ce code n'est pas entré directement. Nous devons d'abord mettre de côté l'ordinateur pendant quelques minutes pour développer un algorithme. Nous commençons par réfléchir à ce que nous devons faire. Nous interprétons la question et identifions les principales étapes différentes : la simulation de la variable aléatoire, le calcul de la moyenne et la création d'un graphe. Ce qui se passe après est un peu difficile à décrire. En général, nous écrivons sur un papier ce que nous devons faire en mots, nous cherchons dans la bibliothèque de *Mathematica* les fonctions préprogrammées dont nous avons besoin et nous traduisons partiellement les étapes en utilisant certains aspects du langage de *Mathematica*. Nous finissons toute la traduction sur l'ordinateur, en écrivant le code. Nous sommes alors prête à calculer. Notons que la façon dont le professeur pose les questions fournit certains indices de ce que nous devons voir et donc, nous avons aussi une façon de vérifier notre petit code : une courbe qui ne tendrait pas vers zéro nous paraîtrait suspecte et nous conduirait à revoir le code.

Nous commençons avec le premier graphique demandé par Paul. En regardant le comportement de la ligne, nous nous demandons : est-ce que ça converge vers 0 ? Peut-être,

[5] Pour des raisons de simplicité, nous avons choisi d'utiliser notre environnement de programmation préféré, *Mathematica*, au lieu de celui utilisé par les étudiants de Paul. Nous estimons que ce changement de langage ne change pas de façon importante notre compréhension de l'exploration par l'étudiant rendue possible par la programmation.

mais nous n'en sommes pas certaine. Heureusement, le fait que nous ayons écrit le programme nous permet de modifier rapidement le code pour produire plusieurs autres graphiques. Nous ajoutons d'abord 10 000 simulations supplémentaires et faisons un autre graphique, qui s'étend sur plus large, de la fonction « runningaverage » (ou « moyennemobile »). Le graphique qui apparaît par défaut ne suffit toujours pas à nous convaincre. Alors nous jouons avec les limites de l'affichage des axes jusqu'à ce que nous trouvions un graphique qui nous plaît (voir la suite des images à la Figure 4.26 ci-dessous). L'évolution de ces trois graphiques constitue un

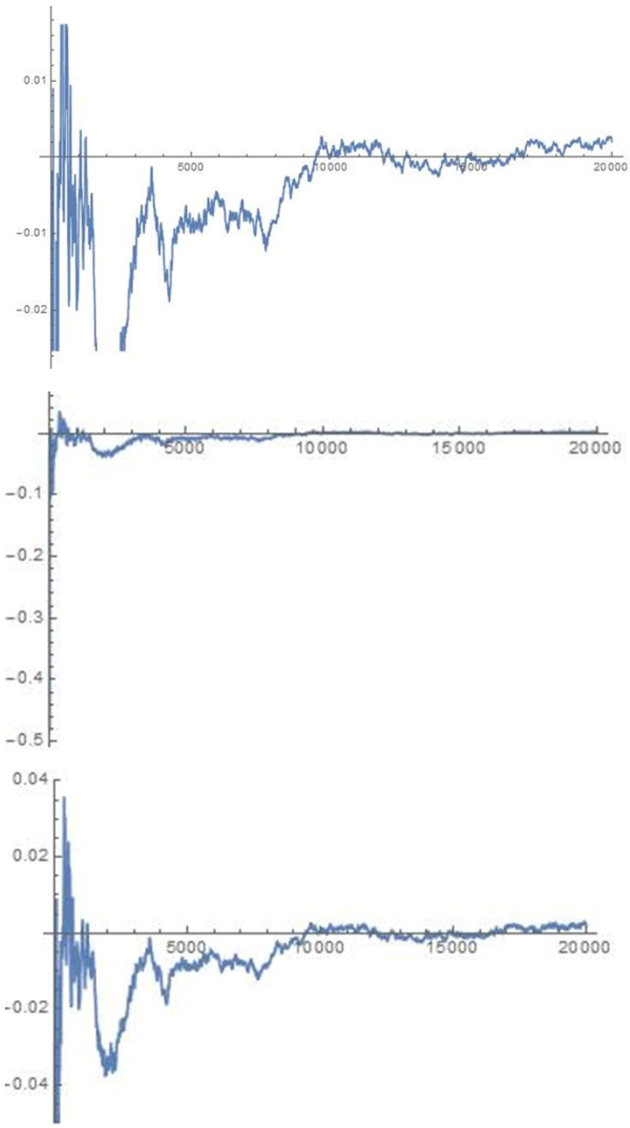


Figure 4.26 La recherche d'une image convaincante :
le graphe par défaut (en haut), le graphe de tous les points (au milieu)
et le graphe qui nous plaît le plus (en bas).

exemple très semblable à celui décrit par Artigue (2002) quand elle introduit la complexité de la genèse instrumentale d'une calculatrice graphique. Dans notre cas, même si nous savions comment programmer en *Mathematica,* nous avons dû apprendre à mieux utiliser l'option « PlotRange » qui peut être ajoutée à la fonction préprogrammée « Plot ». Nous avons donc progressé dans notre genèse instrumentale avec *Mathematica.* À un moment qui est difficile à déterminer, nous finissions par reconnaître aussi l'importance de l'axe des y. Vers la fin de la ligne bleue, nous semblons toujours être très proche de zéro. Donc, oui, il semble bien que notre fonction tend vers zéro après tout.

L'écriture du code pour la deuxième question va plus rapidement en raison de l'expérience gagnée avec la première. Nous arrivons au code suivant :

$x2[y\_] := Tan[y];$
(*définir la variable*)
$ydata = Table[RandomReal[\{(-Pi) \,/\, 2, Pi \,/\, 2\}], \{i, 1{,}20000\}];$
(*simuler 20 000 chiffres distribués uniformément sur l'intervalle (-π/2, π/2)*)
$runningaverage2[n\_] := 1 \,/\, n * Sum[x2[ydata[[i]]], \{i, 1, n\}];$
(*calculer la moyenne mobile*)
$Plot[runningaverage2[n], \{n, 1{,}10000\}]$
(*faire un graphique de la moyenne mobile*)

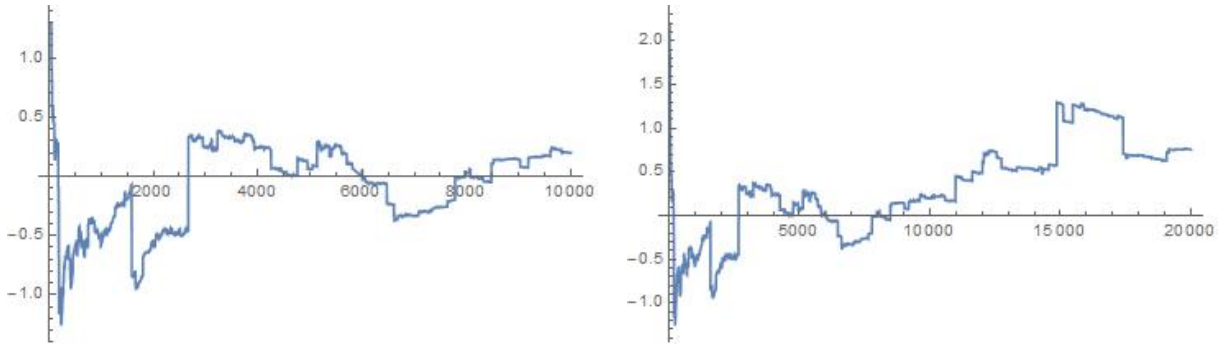


Figure 4.27 La moyenne mobile de la simulation de la variable $X = tan\ Y$.
La représentation sur l'intervalle [0, 10 000] (à gauche) et [0, 20 000] (à droite).

Cette fois-ci, notre code génère un graphique dont le comportement semble bizarre ! Est-ce que la suite des moyennes mobiles converge vers 0 ? Comme précédemment, nous utilisons notre connaissance du langage de *Mathematica* pour aller plus loin. Si nous avions des doutes

en regardant le graphique à gauche à la Figure 4.27, celui à droite nous convainc que la ligne bleue ne semble aller « nulle part en particulier ». Plus précisément, nous remarquons que la suite des moyennes semble converger vers zéro à plusieurs reprises, mais par la suite on observe un grand saut qui éloigne la courbe de l'axe des x. Pourquoi ?

Validation d'une propriété ou démonstration d'un théorème (T3)

Comme en témoigne l'énoncé, l'étudiant, au moment de compléter les problèmes posés par Paul, n'a pas à aller plus loin que la phase d'exploration décrite ci-avant. On ne lui demande pas de dégager et d'expliquer les différences entre les deux situations explorées, même si quelques étudiants pourraient être en mesure d'y arriver. Mais évidemment, les explorations que nous venons de faire sont intimement liées à un résultat important en mathématiques : elles nous montrent la signification et les limites d'application de la loi des grands nombres. En répondant à la première question, nous observons un exemple où ce théorème est valide, tandis que l'exploration exigée par la deuxième question nous montre une instance où la loi ne peut pas être appliquée. Il est à prévoir que les étudiants reçoivent cette explication et tous les théorèmes et preuves correspondants à un certain moment du cours, avant ou après l'exploration. En fait, dans le devoir suivant, ils doivent rédiger une preuve formelle pour expliquer les observations qu'ils font dans cette exploration. Mais pour le moment, ils n'ont pas à penser en ces termes de validation formelle, et ils ne le font certes pas spontanément ; certains d'entre eux semblent attribuer aux résultats dégagés de l'exploration à l'aide de l'ordinateur une force de vérité supérieure à celle de la preuve. Paul explique :

> One student said, "I really liked the *Octave* problems because they showed me that the theorems were true." This is really the opposite point of view from what a mathematician takes. The mathematician sees the truth in the theorem, right? You prove the theorem, then okay it's true, that's it, I don't need to do any experiment, I've already proved the theorem.

Tandis que la preuve est le but ultime et nécessaire d'un mathématicien dans sa recherche, les étudiants ne ressentent pas toujours le besoin d'une preuve pour croire aux théorèmes. Nathan, dans sa présentation des images du mouvement Brownien (voir la page 86), préfère guider ses étudiants dans la compréhension des théories sans leur montrer les définitions, théorèmes et preuves formelles correspondants, qu'il considère trop lourds pour

eux au moment où ils suivent typiquement son cours. Mais, si une preuve n'a pas toujours le même poids pour l'étudiant que pour le mathématicien, l'exploration faite par l'étudiant dans un exemple comme celui de Paul pour comprendre les théorèmes est très représentative de ce qu'un mathématicien ferait avant de découvrir une propriété intéressante et de vouloir la démontrer. Philippe, qui guide ses étudiants dans une exploration presqu'identique à celle de Paul, explique :

> On se pose des questions avant d'essayer de démontrer des choses. C'est comme ça que ça fonctionne dans la recherche. [...] on essaie de voir ce qui se passe avant. Et l'erreur que les étudiants font souvent ici, c'est d'essayer de démontrer quelque chose sans comprendre avant ce qui se passe. Ils mettent tout de suite les epsilons... Et c'est la pire chose à faire.

Philippe n'est certes pas le seul à mentionner les difficultés, éprouvées par les étudiants de premier cycle, à faire des preuves des propositions mathématiques. Barbara suggère même que les étudiants ont peur de la preuve. Or, la démonstration de théorèmes est une tâche très fréquente dans la formation d'un étudiant en mathématiques. On peut donc se demander si les difficultés que les étudiants ont avec la preuve viennent du fait qu'ils passent l'essentiel de leur temps d'apprentissage à recevoir des résultats lisses et polis plutôt qu'à les dégager à partir d'explorations qu'ils auraient faites eux-mêmes, ou sous la direction du professeur. Un des mathématiciens s'exclame : « I still don't believe that formal proof is a reasonable way to be learning math. It's something you do a second or a third time through a subject, not the first time. » Selon Anthony, il faudrait révéler le vrai processus d'exploration vécu par le mathématicien avant d'en arriver aux preuves formelles. Il n'exige pas pour autant que chaque théorème soit précédé d'une exploration, tout comme il ne souhaite pas non plus éliminer de la formation cette tâche de validation si importante en mathématiques. Comme l'explique Paul, il y aurait beaucoup à gagner à faire vivre aux étudiants ces deux volets (T2 et T3) de la recherche mathématique :

> When [the students] put the two together [(the explorations and the theorems)] then there's some kind of synergy that takes place where each one informs the other one. And it's a much more educational experience for them: they really learn a lot more.

Dans tous les exemples décrits dans cette section, le but principal poursuivi par le professeur lorsqu'il programme pour son enseignement ou lorsqu'il fait participer ses

étudiants à l'activité de programmation est de les aider à s'approprier des théories mathématiques abstraites. En fait, l'importance supérieure accordée à ce premier enjeu est souvent invoquée par ces professeurs pour justifier de ne pas exiger une interaction plus riche de leurs étudiants avec la programmation. Ainsi, Philippe explique pourquoi il choisit de créer des programmes informatiques lui-même plutôt que de demander à ses étudiants de s'engager dans des activités de programmation : « Le but c'est de se familiariser avec le raisonnement mathématique au niveau des probabilités. [...] donc ça reste des mathématiques pures. [...] ça ne fait pas partie du cours que l'étudiant soit capable de faire une simulation et des choses comme ça. » Même Paul, qui planifie plusieurs activités de programmation pour ses étudiants explique : « In the end we want to test them on probability. The purpose of the programming is to help them learn probability. » Il est intéressant de noter que les mathématiciens purs ne parlent presque jamais du potentiel de telles activités pour aider leurs étudiants à développer des compétences qui pourraient leur être utiles s'ils devenaient chercheurs, en apprenant à définir, à représenter, à manipuler et à explorer des objets mathématiques. Nous reviendrons sur cette idée dans les prochaines sections. Il convient tout de même de souligner le potentiel de certaines activités que nous avons décrites, comme celle de Paul, qui, en plus d'aider les étudiants à comprendre le sens des conditions d'un théorème important, contribue aussi, par la mise à contribution de générateurs de nombres aléatoires, au développement chez les étudiants de compétences de modélisation et de simulation (E2).

4.3.2 Le développement de compétences de modélisation et de simulation

Ce deuxième enjeu ne reçoit pas toujours une attention aussi grande que le premier dans tous les départements de mathématiques. Ben explique : « What's missing in a lot of mathematics programs, my feeling, is the connection to modelling, simulation, and computing. We are not harnessing that in our math programs [...] the power of computers to solve mathematical problems. » Selon ce mathématicien, les étudiants devraient au moins avoir le choix de s'inscrire dans un programme qui pourrait, par les compétences qu'ils y développeront, faire de leur baccalauréat non seulement un ticket d'entrée à des études supérieures, mais aussi une formation qui pourrait être recherchée dans des emplois variés. Il spécifie : « In no way do I want that to replace all the traditional branches. I just want to add it as a possibility, for some people to follow. » Quand ce deuxième enjeu fait partie des curricula

en mathématiques au niveau universitaire, il est typiquement réservé à des groupes très spécifiques d'étudiants : ceux qui décident de se spécialiser en mathématiques appliquées, en statistique ou en actuariat. Plusieurs mathématiciens de notre étude questionnent cette séparation en soulignant le potentiel de E2 pour exposer tout étudiant à un plus vaste éventail d'activités mathématiques possibles et pour leur ouvrir un maximum de portes après leur baccalauréat. Ainsi, Nathan suggère :

> Il faut transmettre une connaissance générale en mathématiques [...] aux étudiants qui sortent avec le diplôme de bac. Mais en même temps il ne faut pas les déconnecter complètement de la réalité. [...] je ne dis pas qu'il faut former des gens de façon professionnelle pour des emplois bien précis, mais il faut leur donner des outils pour être de bons ambassadeurs des mathématiques. [...] j'ai vu des collègues tomber au combat, des gens qui arrêtent après le bac, la maîtrise ou même le doctorat, puis finalement, quand vient le temps de trouver un emploi, quand vient le temps de réaliser c'est quoi la vraie vie [...] ce qu'on peut faire avec des mathématiques [...] on ne sait même pas si c'est possible de faire autre chose avec nos mathématiques.

Traitement de données issues de situations réelles (T4)

La quatrième tâche sur notre liste en est une qui pourrait aider à « connecter » les étudiants à la réalité, au sens où l'entend Nathan. Comme nous l'avons abordé dans notre discussion des pratiques en recherche (voir la section 4.2.1), l'utilisation des outils informatiques se révèle essentielle pour la manipulation de données (réelles ou simulées) concernant un phénomène réel et leur représentation (souvent sous forme graphique) pour en dégager des régularités.

Un exemple fourni par l'un de nos participants est particulièrement illustratif des possibilités. Ben, comme plusieurs autres mathématiciens de l'étude, enseigne des cours dans lesquels on aborde le mouvement Brownien. Tout comme Nathan (voir la page 86), Ben intègre une exploration de représentations graphiques par la programmation. Mais une des grandes différences entre les deux explorations est que celle de Ben s'est orchestrée dans un environnement de laboratoire où les étudiants ont à écrire eux-mêmes de petits programmes en *Excel* (N4) pour générer la série des graphes. De plus, contrairement aux autres mathématiciens enseignant le mouvement Brownien, Ben insiste sur la façon dont il mène ses étudiants au-delà de la simple compréhension de la théorie (E1). Il fait explicitement le lien

entre le mouvement Brownien et le marché boursier, un phénomène auquel il s'intéresse dans sa propre recherche :

> I'm trying to sensitize them to the fact that in random walks, you see this behaviour where it looks like a trend, where it looks like a big drop, but there's nothing going on. And it's the same as when you're looking at stocks. You see a stock going really really high or really really low; it's mostly meaningless. It's just a random walk.

Comme dans plusieurs exemples déjà décrits, le professeur cherche à ébranler l'intuition des étudiants et à les mettent en déséquilibre. Ben veut que ses étudiants apprennent qu'il n'est pas possible de prédire les retours futurs des bourses sachant le comportement précédent. Mais au lieu de se contenter de le leur dire, il les pousse à s'en convaincre eux-mêmes en traitant de vraies données issues des marchés boursiers. Il justifie son approche :

> I'm trying to teach them not only the mathematics […] But also to give them the feeling. The feeling of what happens. It's not just to compute the formulas. It's to actually see with real data. Several people in this class have said this is the first time in their university career that they've worked with real data. The first time. […] It's unbelievable. […] with me, they're working with real data every week.

Si l'écriture de programmes très simples dans ce cas a une valeur *pragmatique* (le traitement des données est beaucoup plus efficace en *Excel*), Ben met toujours l'emphase sur sa valeur *épistémique*. Dans son cours, chaque étudiant choisit un titre sur le marché boursier et suit son comportement à travers le trimestre. Chaque semaine, les étudiants font des calculs différents selon les instructions du professeur et discutent de leurs résultats. Ce faisant, ils trouvent, par exemple, que la corrélation entre les retours passés et actuels est presque nulle, ce qui pourrait les convaincre enfin de la difficulté de prédire le comportement de ces titres. Mais en plus, ces étudiants sont exposés aux aspects du processus de modélisation.

Dans les autres exemples décrits par nos participants, c'est plutôt le mathématicien qui traite des données, ce qui confine à nouveau l'étudiant à un rôle d'observateur de résultats de programmes (N0). Un exemple de Philippe nous permet d'imaginer d'autres occasions où les étudiants en mathématiques pourraient être exposés à la collection et au traitement de données. Dans un de ses cours, ce professeur emploie un programme en *Mathematica* pour expliquer « la ruine du joueur ». En particulier, il cherche à montrer à ses étudiants l'impact énorme d'un léger biais dans un jeu de casino :

Situation : On arrive au Casino avec 100$ pour jouer un certain jeu. Dans ce jeu on mise un dollar chaque fois. Si on gagne, on obtient un dollar en plus du dollar qu'on a misé. Sinon, on perd le dollar.

La question est alors : si chaque fois la probabilité de gagner vaut p, quelle est la probabilité d'atteindre 200$ (de doubler son montant initial) avant de tout perdre (de se ruiner) ? Au lieu d'aller directement au calcul de cette probabilité en fonction de p, Philippe essaie de donner une idée de ce qui pourrait se passer en simulant le jeu et en représentant les résultats sous forme graphique. Les graphiques suivants, à gauche et à droite, montrent la fortune d'un joueur respectivement après 40 000 essais, avec une probabilité de succès de 1/2, et 4000 essais, avec une probabilité de 9/19.

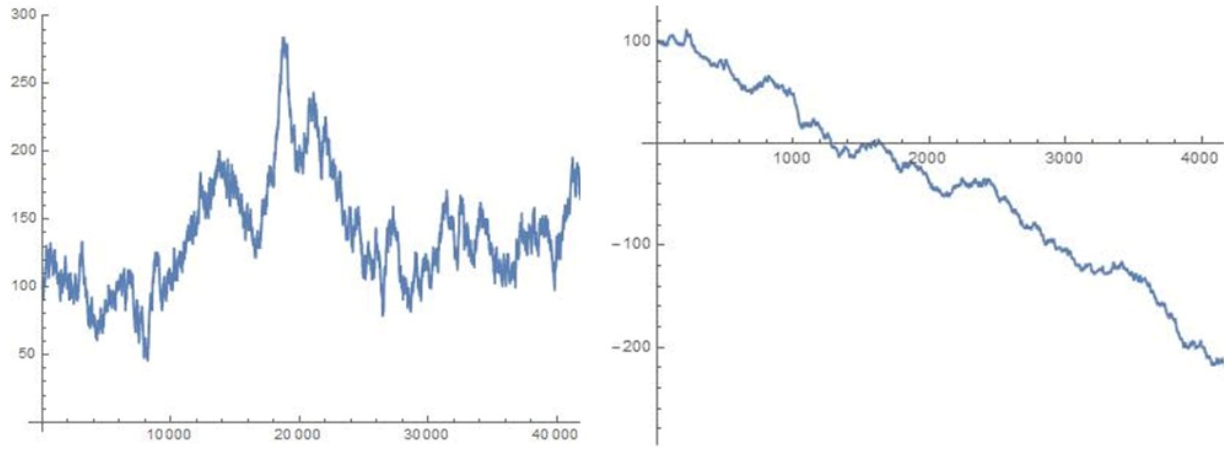


Figure 4.28 La fortune d'un joueur après x essais pour la probabilité 1/2 (à gauche) et 9/19 (à droite) de succès.

En comparant le comportement dans ces deux graphiques, on peut formuler la conjecture que la probabilité de se ruiner est très grande même avec un très léger désavantage. Philippe confirme cette observation en procédant aux calculs des probabilités à partir des formules connues. De son discours justifiant son choix de laisser les étudiants en position d'observation (N0), il est clair que Philippe veut davantage motiver les étudiants à atteindre E1 (l'appropriation de la théorie) ou E3 (l'appropriation de méthodes de calcul) qu'à les aider à développer des compétences de modélisation. Mais il ajoute : « Ça me permet un peu de lier la théorie à la vraie vie. Idéalement, j'irais au casino. Mais, c'est le plus proche que je peux avoir

de la réalité. » Les capacités de simulation en *Mathematica* permettent de générer des données rapidement, de les traiter devant les étudiants et d'établir des liens avec des situations réelles intéressantes, sans avoir à organiser une « sortie scolaire » au casino (et ruiner les étudiants). On peut toutefois se demander si la visualisation des résultats de simulation a le même potentiel que l'occasion de faire de véritables expériences.

Pour Alain, un professeur dans un département de génie, il est essentiel que ses étudiants fassent de véritables expériences et en analysent ensuite les résultats, possiblement à l'aide des programmes informatiques. Il explique : « La mécanique des fluides, ça n'est pas une abstraction numérique. Oui, il y a la théorie qui décrit nos équations, qui décrit la statique, qui décrit la dynamique. Mais dans une école de génie, il faut leur donner des démonstrations avec une expérience. » Alain justifie souvent ses choix d'enseignement en référant au fait que ses étudiants seront un jour ingénieurs ; ils apprennent à collecter et traiter des données pour être en mesure un jour de développer de nouveaux modèles. En employant la même logique, il est raisonnable de croire qu'un étudiant en mathématiques pourrait profiter du développement de mêmes compétences, autant pour apprendre à modéliser un phénomène que pour se préparer à travailler avec des collègues d'autres disciplines. Cela dit, l'exposition d'un étudiant à la collecte, la manipulation et l'analyse de données réelles semble moins une priorité en mathématiques qu'en génie.

Recherche des solutions ou expérimentation avec un modèle (T5)

Tout comme les étudiants de premier cycle ne passent pas beaucoup de temps à développer leurs propres théories mathématiques, ils développent peu de nouveaux modèles. Lorsqu'ils développent certaines compétences de modélisation et de simulation, ils le font typiquement en explorant des modèles mathématiques existants. Seulement quelques participants de notre étude décrivent des exemples concrets où leurs étudiants explorent des modèles à l'aide de la programmation. L'accomplissement de cette tâche, dans les exemples fournis par nos participants, peut impliquer les sous-tâches suivantes : la programmation du modèle, la recherche de résultats spécifiques, l'expérimentation avec le modèle et l'interprétation des résultats dans le monde réel. Remarquons que chacune de ces sous-tâches correspond à une tâche dans les pratiques des mathématiciens appliqués (voir la section 4.2.1).

Il faut noter quand même que l'étape de validation, qui est si importante en recherche, n'est pas souvent présente dans les activités d'apprentissage proposées par nos participants.

Un exemple décrit par Norman illustre la façon dont les étudiants pourraient réaliser ces sous-tâches qui ressemblent à l'activité mathématique des chercheurs de notre étude. Dans un cours des cycles supérieurs portant sur la modélisation, ce professeur prépare une activité de laboratoire concernant le système suivant, qui modélise la population de deux types d'animaux, X et Y, en fonction du temps t :

$$\frac{dX}{dt} = X(a + bX + cY)\,;$$

$$\frac{dY}{dt} = Y(d + eY + fX).$$

Avant d'explorer ce système, les étudiants comparent la précision de deux méthodes numériques qui pourraient être employées pour le résoudre, ce qui peut constituer une étape importante de *la programmation du modèle*. Pour ce qui est du développement des algorithmes, Norman s'en charge presque complètement : il identifie les variables et les paramètres importants, les différentes étapes de l'algorithme, la façon de les implanter en *Excel* et même la structure pour l'output. Mais c'est l'étudiant, possiblement avec un partenaire, qui doit lire et suivre les instructions pour créer le programme (N4). L'étudiant est invité à vérifier le fonctionnement du programme sur un exemple simple (en prenant $a = 2$ et $b = c = d = e = f = 0$) en même temps qu'il compare les deux méthodes numériques ; cela peut lui donner la confiance nécessaire pour commencer les calculs plus complexes. Il est intéressant de noter que Norman suggère à ses étudiants d'utiliser toujours, pour leurs explorations, 0,1 comme valeur pour la taille du pas de temps Δt, car il sait que cette valeur donnera des résultats raisonnables. De toute façon, il les encourage à valider leurs résultats en apportant de petits changements à cette valeur et en vérifiant qu'ils n'observent aucun changement significatif dans les résultats.

Une fois que les étudiants ont complété la programmation du modèle en *Excel*, ils peuvent entrent dans l'étape *de recherche des solutions et d'expérimentation*. Norman leur demande d'essayer plusieurs valeurs pour les paramètres dans le modèle et d'en observer l'effet sur le comportement des solutions. Pour aider à circonscrire leurs premières explorations, il

suggère de prendre $a = d = 1$ et $b = e = -1$ et de varier c et f sur des valeurs négatives différentes. Il leur indique de plus : « In some cases, one species gets driven to extinction; in other cases, the two species can coexist. » Il les encourage aussi à jouer avec les points initiaux pour voir s'ils ont un effet sur le sort des populations.

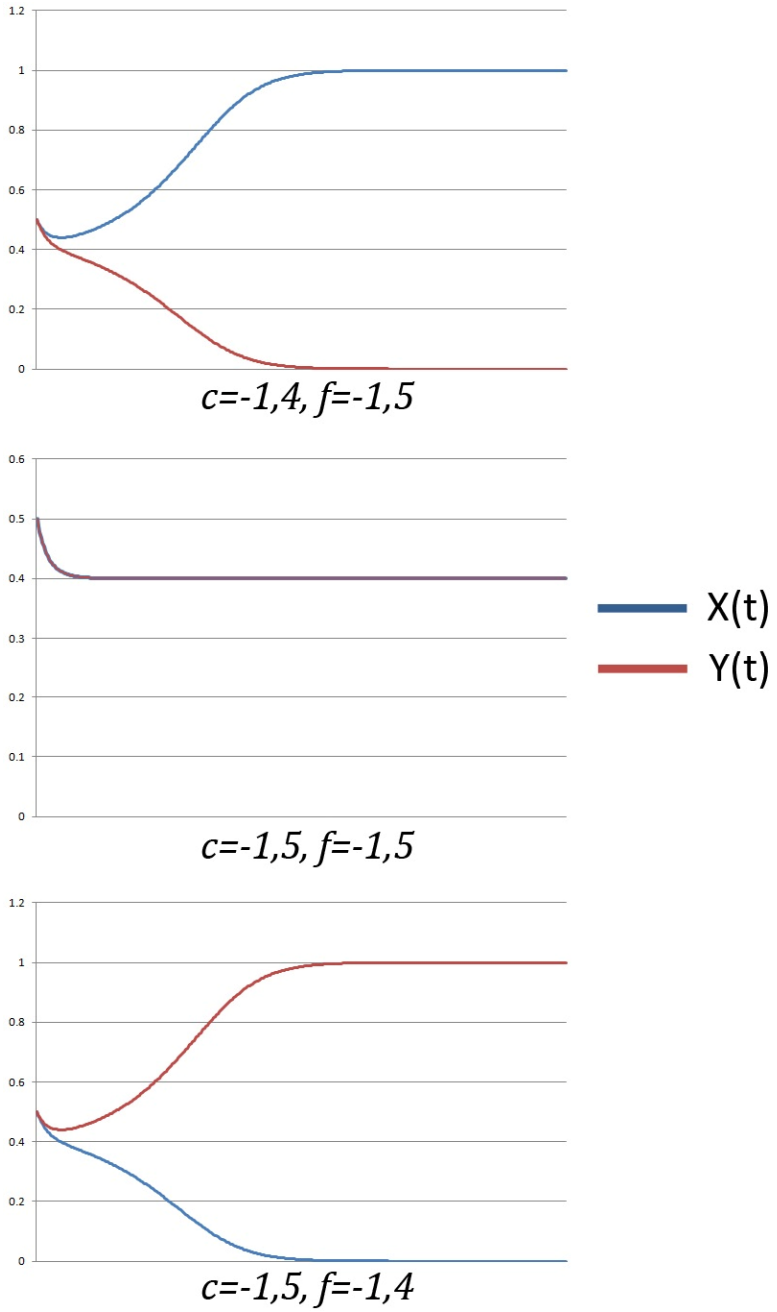


Figure 4.29 Le comportement du système à travers le temps pour des valeurs différentes des paramètres c et f.
$X(0) = Y(0) = 0,5$, $a = d = 1$, $b = e = -1$.

Avec telles indications, il est probable que l'exploration de chaque étudiant (ou paire d'étudiants) serait unique. Nous pouvons quand même indiquer le type d'observations que les étudiants pourraient faire. Par exemple, supposons que nous commencions avec un cas très simple où il y a au début le même nombre d'animaux pour chaque espèce. En variant les valeurs de c et de f, un étudiant pourrait remarquer que quand les valeurs des paramètres sont les mêmes, les deux espèces peuvent coexister et leurs populations convergent vers la même valeur (deuxième graphique à la Figure 4.29). Par contre, si la valeur de c est inférieure à la valeur de f, l'espèce représentée par X devient extincte, tandis que l'espèce représentée par Y survit et approche la valeur de 1 (troisième graphique à la Figure 4.29). Si cette relation ne se maintient pas toujours lorsqu'on change les valeurs initiales, ces types d'observation permettent d'anticiper comment les valeurs de c et de f sont liées aux valeurs des populations : par exemple, plus c devient fortement négatif, plus cela a un impact négatif sur la valeur de X. Ces relations sont facilement explicables en revenant aux équations qui forment le système à l'étude et en y examinant la place et le rôle des paramètres.

Quand on change les valeurs initiales de façon telle qu'elles ne sont pas les mêmes pour les deux espèces, les changements au comportement observé peuvent être radicaux. Par

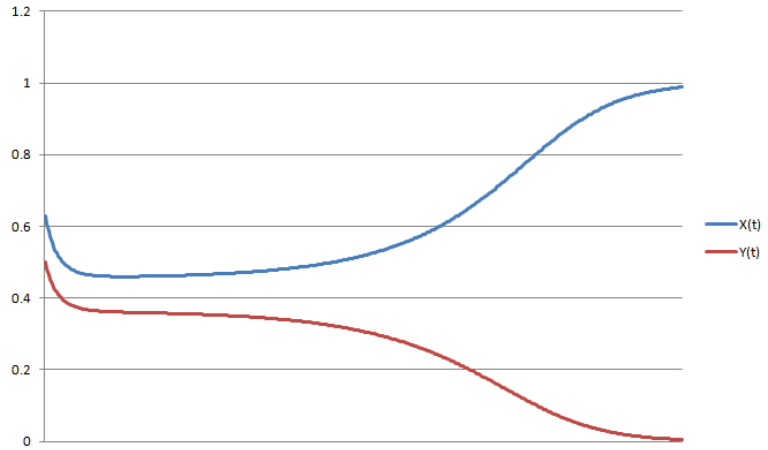


Figure 4.30 Changement de $X(0) = 0,5$ à $X(0) = 0,63$, avec le troisième graphique à la Figure 4.29 comme point de départ.

exemple, dans une situation comme celle montrée dans le troisième graphique à la Figure 4.29, on peut réussir à échanger le sort des deux populations en augmentant suffisamment la valeur de $X(0)$ (Figure 4.30). Les étudiants pourraient découvrir enfin une autre idée

importante, mais assez intuitive : plus la population initiale d'une espèce est grande, plus cette population a une chance de survivre. Évidemment, cet énoncé est trop général pour représenter tout le comportement complexe du système. Mais en le combinant à d'autres observations faites, les étudiants peuvent au moins avoir une idée de ce qui se passe.

Une autre question donne aux étudiants l'occasion de devenir plus autonomes dans leurs explorations. Norman spécifie certaines conditions : il leur dit de poser $b = e = 0$ et de s'assurer que a et f sont positifs et que c et d sont négatifs, ce qui résulte en un modèle simple de proie-prédateur. Mais cette fois-ci, les étudiants doivent déterminer eux-mêmes le comportement du modèle et *l'interprétation des résultats dans le monde réel*. Une différence qu'ils vont remarquer ici est que les solutions du système ont un caractère périodique, pour autant que les points initiaux soient supérieurs à zéro. Autrement dit, les populations montent et descendent continuellement. Un exemple de ce comportement, avec $\Delta t = 0{,}06$, est montré à la Figure 4.31 ci-dessous. Pour interpréter ce graphique dans le monde réel, les étudiants doivent se rappeler que le système représente l'interaction entre un prédateur et sa proie. Quand il y a beaucoup de proies, le prédateur a beaucoup de nourriture et donc il peut se reproduire facilement, ce qui cause sa population à augmenter. Mais plus il y des prédateurs, plus les proies sont tuées. Éventuellement, la population des proies devient si petite qu'elle ne soutient plus les besoins des prédateurs ; les prédateurs commencent à mourir aussi. Moins de prédateurs implique que c'est maintenant la proie qui peut facilement se reproduire. Ainsi le cycle recommence.

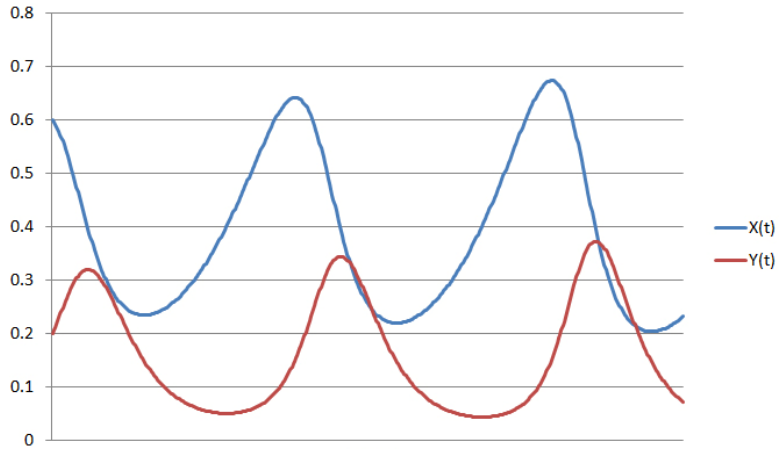


Figure 4.31 Un exemple du comportement périodique dans le modèle de proie-prédateur.

Remarquons que ce graphique présente une quasi-périodicité plutôt qu'une périodicité exacte. Cette exploration pourrait donner lieu ainsi à une discussion riche qui mettrait à contribution à la fois les connaissances mathématiques des étudiants, leurs compétences de modélisation et de simulation et leur genèse instrumentale. Les solutions dans ce cas sont-elles vraiment non périodiques ? La quasi-périodicité montrée par le programme pourrait-elle résulter des contraintes induites par l'outil ? Comment pourrait-on le vérifier ? Pour ce modèle de proie-prédateur, associé à Lotka et Volterra, on sait que les solutions montrées ci-avant sont périodiques. Ainsi la quasi-périodicité qu'on observe résulte du choix du pas de temps, de la précision de la méthode d'intégration utilisée et du cumul d'erreurs numériques. On peut se demander si cette activité pourrait être enrichie par l'ajout d'*une validation* rigoureuse de la périodicité des solutions, comme le ferait un vrai mathématicien appliqué.

Même si l'activité de Norman se situe dans un cours de niveau maîtrise, il ne serait pas impossible que les étudiants au niveau baccalauréat puissent participer à des activités semblables. En effet, Ben donne aux étudiants de premier cycle à son université plusieurs devoirs dans lesquels ils écrivent des programmes (N4) pour explorer des modèles tels celui de l'exemple de deux populations ci-avant. L'ensemble des exemples décrits par Ben illustre la variété des expériences qu'un étudiant pourrait vivre afin de développer des compétences de modélisation et de simulation. Par exemple, dans un devoir, ses étudiants analysent des séquences d'ADN, qui sont représentées par des suites de longueur n constituées de quatre lettres, chacune associée à un nucléotide différent. Les étudiants écrivent des programmes pour chercher des résultats très spécifiques : par exemple, étant donné deux séquences d'ADN de longueur n, quelle est la plus longue sous-suite commune ? Ils doivent aussi expérimenter pour répondre à des questions comme : la position d'un nucléotide par rapport aux autres est-elle aléatoire ? Ou y a-t-il des régularités notables dans les séquences ?

Pour répondre à ces types de question, l'utilisation de l'ordinateur apparaît incontournable pour des raisons *pragmatiques* : cet outil permet des calculs qui seraient trop difficiles et trop longs pour faire autrement. En plus de cela, les capacités d'organisation et de visualisation supérieures des outils informatiques ainsi que la rapidité des solutions facilitent des expériences qui peuvent mener l'étudiant à mieux comprendre les objets impliqués (valeur

épistémique). Mais on peut à nouveau se demander, comme nous l'avons fait à la section 4.3.1, si l'étudiant devrait créer lui-même les outils informatiques qu'il utilise.

Programmer ou ne pas programmer, telle est la question

Barbara est une mathématicienne qui doute de la pertinence d'exiger de ses étudiants qu'ils programment. Dans son cours de biologie mathématique, elle remarque que ses étudiants ont beaucoup de difficultés avec *MATLAB ;* alors, elle décide de leur fournir des programmes pour explorer les modèles qu'elle enseigne. Par exemple, elle décrit un moment où elle donne aux étudiants un programme qui crée un champ de vecteurs pour un certain modèle pour qu'ils puissent en analyser le comportement. Les étudiants font des manipulations au niveau de l'interface (N1) et ils ont accès au code (N2), mais ils ne sont pas du tout actifs dans l'activité de programmation. Barbara justifie son approche en disant : « It wasn't so much how to program a vector field, it was how to use a vector field to understand the model. » Elle attribue ainsi une importance plus grande à l'appropriation de concepts ou de techniques de modélisation (E1 et E2) qu'au développement de compétences de programmation (E4). Le choix de Barbara d'offrir au moins à ses étudiants l'occasion de travailler avec des programmes déjà développés lui semble être un bon compromis dans sa situation ; ses étudiants peuvent encore se familiariser avec un aspect important de l'activité mathématique.

D'autres mathématiciens sont plutôt d'avis qu'il y a beaucoup à gagner en faisant grimper au niveau N4 d'interaction des étudiants avec la programmation. En écrivant un programme, Ben explique que l'étudiant comprendra mieux ce qu'il est en train de faire : « It's very hard to write a program and not understand what it's doing. You know, it's a different level of comprehension. » Comme comparaison, il décrit la réaction des étudiants à un moment où il leur demande de manipuler des programmes déjà préparés : « The comments from the students have always been saying, you know, it was just typing. It was an exercise in typing. They really didn't know what it was doing or why it was doing it. » On peut imaginer que les étudiants seront aussi perplexes s'ils restent en position d'observation des résultats d'un programme écrit par leur professeur (N0). Par contre, avec une meilleure compréhension du programme parce qu'il en aurait écrit de grands pans, l'étudiant serait en mesure de le manipuler, de le modifier selon ses besoins et de gagner ainsi en autonomie dans le processus

d'instrumentalisation de l'ordinateur. Alors la programmation des outils de simulation pourrait donner à l'étudiant une certaine puissance et plus de contrôle sur ses explorations, initiées soit par le professeur, soit par lui-même. Rappelons que quelques participants utilisent cette même idée de contrôle pour justifier leur propre programmation dans leur recherche.

En invitant l'étudiant à programmer, on le met ainsi en contact avec un pan de l'activité de plusieurs mathématiciens. Mais des expériences de programmation peuvent avoir d'autres effets positifs dans le contexte d'apprentissage. Ben parle aussi de l'intérêt de la vérification d'un programme informatique comme une discipline de travail très formatrice. Il dit que « that's a wonderful kind of thinking that we often don't do enough to engender in mathematics. Is your answer reasonable? Does that make sense? » Ce processus de vérification, pourrait-t-il être intégré dans les manières de faire de l'étudiant face à toute situation mathématique ?

Les compétences de modélisation et de simulation au delà des cours « appliqués » ?

En dépit du haut niveau d'interaction avec la programmation des professeurs en mathématiques appliquées de notre échantillon, les expériences qu'ils font vivre à leurs étudiants de premier cycle se situent principalement dans des cours spécifiquement consacrés à l'élaboration, l'utilisation et l'exploration des modèles mathématiques existants. On peut se demander donc s'il y a d'autres moments dans un baccalauréat en mathématiques où l'on pourrait envisager l'intégration de la programmation dans le traitement de données (T4) ou l'expérimentation avec des modèles (T5) pour viser E2.

Un exemple de Barbara nous permet d'imaginer l'inclusion de la programmation dans un cours d'équations différentielles. Cette professeure consacre du temps dans son cours à expliquer des contraintes liées à la *transposition des savoirs dans des environnements informatiques* (Balacheff, 1994, cité dans Trouche, 2000). Elle explique : « We really do need to know the theory, so we can have confidence in our results. Because the computer sometimes lies. » En se basant souvent sur ses propres expériences de recherche, Barbara fournit des exemples spécifiques de moments où les résultats de l'ordinateur peuvent être complètement faux, ce qui pourrait être difficile à déceler si on n'avait pas de connaissances théoriques. Par exemple, parfois on peut prouver que des solutions n'existent pas même si

l'ordinateur semble en produire. La périodicité des solutions pourrait être également difficile à conclure simplement à partir de graphiques générés par l'ordinateur et des approximations numériques qui y ont conduit (nous l'avons vu dans l'exemple à la page 104). Si le but ultime visé par Barbara est essentiellement de motiver l'appropriation de la théorie (E1), ses mises en garde révèlent aux étudiants des contraintes importantes des outils informatiques pour faire des simulations ; ils doivent interpréter avec beaucoup de soin les résultats de l'ordinateur, tout comme ils doivent utiliser avec beaucoup de soin des théories en mathématiques. Ces informations participent à la genèse instrumentale des étudiants ; néanmoins, si leur interaction avec la programmation ne dépasse pas le niveau N0, ils sont moins outillés pour développer leurs compétences d'interprétation, si importantes dans le processus de simulation. Un cours d'équations différentielles pourrait-il être un bon endroit pour rendre actif l'étudiant dans l'activité de programmation ?

Quelques mathématiciens de notre étude sont d'avis que l'intégration d'activités de programmation dans un cours d'équations différentielles serait tout à fait naturelle. Paul, par exemple, parle d'un collègue qui donne ce cours et dont les étudiants utilisent *Excel* pour résoudre des équations. Tout comme Norman, ce professeur veut aider ces étudiants à développer des compétences de simulation (E2). Paul ajoute que la puissance des expériences visuelles fournies par l'ordinateur peut aussi soutenir les étudiants dans leur appropriation de la théorie (E1) : « It's one thing to write down the equation and then solve it; but it's another thing to actually see the solution emerge. » Alain suggère qu'une telle intégration de la programmation est aussi importante pour s'approprier de nouvelles méthodes de calcul (E3). Il justifie ainsi son idée d'intégrer le calcul numérique dans les cours d'équations différentielles :

> Les étudiants ne doivent pas connaître seulement la théorie, ils doivent aussi avoir l'habileté pour son utilisation dans un contexte de calcul. [...] un mathématicien qui ne sait pas calculer, il lui manque quelque chose. [...] ce ne sont pas tous les mathématiciens qui vont devenir des théoriciens. Donc, savoir qu'une solution existe, c'est bien beau. Qu'elle est unique, c'est peut-être encore un peu mieux. [...] Mais, il faut avoir des exemples dans lesquels on applique les différents calculs.

4.3.3 L'appropriation de méthodes de calcul et de représentation

Ce troisième enjeu évoqué par nos participants est, tout comme le premier, au cœur de nombreuses formations en mathématiques. Cela n'est certes pas surprenant puisque les tâches

de calculer et de représenter des mathématiques sous plusieurs formes sont intimement associées au travail du mathématicien. Tous nos participants reconnaissent le rôle clé de l'ordinateur dans l'atteinte de cet enjeu. En effet, en cernant les cours propices à l'intégration de la programmation, ils ont énuméré un grand nombre de cours (calcul, algèbre linéaire, géométrie, probabilités, statistique, modélisation, équations différentielles, systèmes dynamiques, etc.) où l'ordinateur peut être utile pour calculer ou visualiser (une forme de représentation). Ce faisant, ils remettent en question les types de méthode de calcul et de représentation qui dominent parfois dans la formation d'un étudiant en mathématiques.

Anthony partage avec nous sa frustration : « I have failed to convince my colleagues that when we put down communication as one of the goals of the programs, visual communication is one part that must be there. But many of them don't believe that. So they're happy to sort of write equations and talk and wave their hands. » Selon ce mathématicien, les étudiants au niveau baccalauréat devraient avoir beaucoup plus d'occasions, tout au long de leur formation, de travailler avec des représentations multiples des mathématiques. Il soutient qu'en développant leur habileté à utiliser des représentations symboliques, physiques et visuelles, on permet aux étudiants d'être mieux outillés pour aborder des problèmes mathématiques. L'intégration des outils informatiques serait un aspect crucial de cette exposition aux représentations multiples.

D'autres participants affirment que les étudiants passent trop de temps à faire des calculs ou des graphiques à la main, alors que ceux-ci pourraient être faits très facilement avec un outil informatique. Ben réfléchit :

> We have often taught mathematics like a sequence of algorithms that certain kinds of very bright people can memorize and spew off at will, right? They learn the mechanics of solving, you know, the quadratic formula [...] They sit down, they memorize the mechanics of that, they get terrific marks, and they go on to the next level. What level? What is that? That's not mathematics, is it? It certainly can't be mathematics at this time, when machines do all of it.

En employant le même raisonnement, Anthony croit que dans un cours comme celui d'algèbre linéaire, les étudiants pourraient passer beaucoup moins de temps à faire des calculs matriciels à la main. Il s'exclame : « I'm not sure what they learn by finding inverses of four-by-four matrices by hand. Except that they make mistakes. » En fait, plusieurs mathématiciens

soulignent le potentiel d'utiliser des logiciels de calcul symbolique comme *Mathematica* pour permettre à l'étudiant de consacrer ses efforts à d'autres tâches plus riches ou de l'ouvrir au travail avec des matrices beaucoup plus grandes. Comme Paul dit : « We spend a lot of time with 3 by 3 matrices. But out there in the real world, they're 3000 by 3000 [...] So, if you have a computer, you can handle it. And, you can also realize what the difficulties are, and what a normal matrix looks like out there in the real world. It's mainly zeros typically, right? »

La résolution de systèmes d'équations avec ces matrices « réalistes » exige souvent des méthodes numériques très spécifiques, ce qui nous mène à l'autre critique sur les méthodes de calcul qui sont typiquement enseignées. Selon quelques participants, il n'y a pas toujours assez d'emphase sur les méthodes numériques. C'est dans les réflexions de Ben que nous trouvons encore un bon point de départ pour comprendre cette critique. Il explique :

> The culture of mathematics is very much built around concrete formulas [...] And that culture's perpetuated in our classrooms. [...] There's certainly a place for analytic solutions, and they are beautiful and so on. But I think we have to now open our eyes to mathematics being much more than just solving algebraic equations, solving PDEs, solving ODEs... I mean, the world is simply too complex for that.

Cette critique concerne la façon traditionnelle dont les cours d'équations différentielles sont enseignés : en transmettant un livre de recettes qui peuvent être utilisées pour trouver les solutions analytiques à un ensemble limité de systèmes d'équations. Quand Adèle envisage l'inclusion d'activités de programmation dans des cours d'équations différentielles, elle souligne aussi les limites de cette approche par recettes : « C'est très peu [d'équations] qu'on sait résoudre effectivement. Mais pour les autres, qu'est-ce qu'on fait ? » Selon elle, Ben et quelques autres mathématiciens, il est important d'ouvrir les perspectives d'étudiants, pas seulement à la possibilité d'utiliser des méthodes numériques pour faire des calculs qu'on ne peut pas faire analytiquement, mais aussi à la nécessité d'utiliser de telles méthodes. Les réflexions d'Anthony résument bien ces idées :

> Most of the students get to fourth year and wouldn't recognize that if I write an integral sign and some crazy formula and dx, that it can't be done in closed form. Almost no integrals can be done. [...] That's why we got that course on numerical integration! Because that's the way almost everything has to be done. Right? [...] when you can't do it by hand, computers are our back-up for a lot, more and more stuff.

Analyse et exploration d'algorithmes (T6)

Parmi l'ensemble des algorithmes envisageables, ce sont surtout les algorithmes numériques qui ont été évoqués par nos participants. Tout comme l'enseignement des compétences de modélisation et de simulation semble être largement réservé à des cours très spécifiques, les étudiants sont pour la plupart initiés à ces algorithmes dans un cours qui leur est consacré. L'activité de programmation permet d'implémenter les méthodes dans des programmes, qui peuvent ensuite être utilisés pour appliquer les méthodes à des équations spécifiques. Même si l'utilisation de programmes n'est pas absolument nécessaire pour analyser et explorer des algorithmes numériques, elle est considérée comme très importante. Comme le dit Alice : « I think that a course in numerical analysis without computing is not fair. You can teach numerical analysis without computer programming, but it's not fair. »

Dans certains cas, la programmation est complètement prise en charge par le professeur qui utilise le programme pour donner des exemples des méthodes et de leur analyse. Les étudiants restent alors au niveau N0 d'interaction avec l'activité de programmation. Alice explique : « If I'm illustrating an algorithmic issue, or if I want them to see the performance of algorithms, if I want to compare and contrast the Forward Euler, Backward Euler, Runge Kutta 4, then I do write code and then I generate some outputs and then I show them what the outputs look like. » Albert et Alain, qui donnent aussi des cours de méthodes numériques, évoquent des illustrations similaires. Par ailleurs, ces trois professeurs encouragent aussi un contact actif de l'étudiant avec la programmation. Par exemple, ils décrivent des moments où, comme professeurs, ils conduisent en classe l'analyse générale des méthodes (p.e., les preuves des taux de convergence), mais ils demandent aux étudiants d'écrire des programmes (N4) pour observer les grandes idées présentées sur des exemples spécifiques. Alain justifie cette approche en s'appuyant sur la conviction qu' « il y a un renforcement positif à obtenir du fait de prendre un algorithme qui a été construit au tableau [...] dont les caractéristiques ont été élaborées en classe et d'aller ensuite les vérifier. »

Évidemment, l'étudiant peut aussi apprendre des méthodes numériques dans d'autres cours, comme ceux de calcul, d'algèbre linéaire, de probabilités et de mathématiques financières. Anthony a même choisi de présenter la méthode de Sturm, qui est utilisée pour

calculer toutes les racines d'un polynôme, dans un cours d'algèbre abstraite. L'exemple suivant, qui nous vient de Norman, se situe dans un cours de modélisation.

Norman n'avait pas initialement planifié une exploration des algorithmes comme partie de l'activité que nous avons déjà présentée (voir la page 101). Mais en essayant l'activité, il avait reconnu que, dans certains cas, la méthode classique d'Euler ne donnait pas des approximations très précises ; il considéra alors la méthode modifiée d'Euler, qui donna de meilleurs résultats (pour une description de ces deux méthodes, voir Fortin, 2011). Enfin, il a décidé d'ajouter à l'activité un problème qui permettrait aux étudiants de faire la même découverte et par conséquent, de s'approprier de nouvelles méthodes de calcul.

Le problème commence par la programmation sur *Excel* des deux méthodes discutées afin d'obtenir les solutions approchées à un système simple :

$$\frac{dX}{dt} = 2X\,;\;\; X(0) = 1.$$

La solution exacte du système est connue : $X(t) = e^{2t}$. Pour chaque méthode, les étudiants comparent les solutions approximées à la solution exacte pour plusieurs tailles de pas h. La figure ci-dessous montre une feuille de calcul en *Excel* qui pourrait être utilisée pour comparer les approximations à la solution exacte quand la taille du pas est égale à 0,1. Le programme est simple ; une fois encore, on peut voir l'intérêt d'un outil aussi répandu qu'*Excel* pour permettre aux étudiants d'explorer des objets et des idées mathématiques.

	A	B	C	F	G	H	I
1		MÉTHODE D'EULER		MÉTHODE MODIFIÉE		VALEUR EXACTE	h
2	t(k)	X(k)	ERREUR	X(k)	ERREUR	X(t)	0.1
3	0	1	0	1	0	1	
4	0.1	1.2	0.021402758	1.22	0.001402758	1.221402758	
5	0.2	1.44	0.051824698	1.4884	0.003424698	1.491824698	
6	0.3	1.728	0.0941188	1.815848	0.0062708	1.8221188	
7	0.4	2.0736	0.151940928	2.21533456	0.010206368	2.225540928	
8	0.5	2.48832	0.229961828	2.702708163	0.015573665	2.718281828	
9	0.6	2.985984	0.334132923	3.297303959	0.022812964	3.320116923	
10	0.7	3.5831808	0.472019167	4.02271083	0.032489137	4.055199967	
11	0.8	4.29981696	0.653215464	4.907707213	0.045325212	4.953032424	

Figure 4.32 Feuille de calcul en *Excel* pour comparer deux méthodes numériques.

Dans l'une des questions, Norman demande aux étudiants d'examiner l'erreur des deux approximations quand $t = 0{,}5$. Plus spécifiquement, il leur demande de comparer la taille de

l'erreur pour plusieurs valeurs de h (p.e., $h = 0{,}1\ ;\ 0{,}05\ ;\ 0{,}025\ ;$ ...) et il leur donne aussi l'indice que l'erreur devrait être proportionnelle à h^p pour un p qu'ils devront estimer. Même avec seulement cinq valeurs de h (0,1 ; 0,05 ; 0,025 ; 0,0125 et 0,00625), les graphiques montrent clairement le comportement attendu :

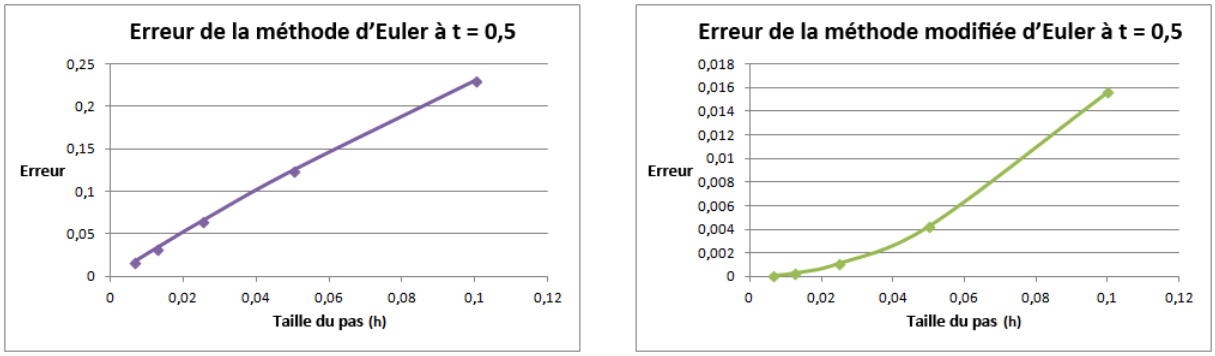


Figure 4.33 Comparaison de l'erreur des méthodes à $t = 0,5$ pour $h = 0,00625\ ;\ 0,0125\ ;\ 0,025\ ;\ 0,05$ et $0,1$.

Norman souhaite que, de ces graphiques simples, sans une analyse de régression avancée, ses étudiants dégagent les régularités suivantes :

1) quand on diminue la taille du pas, l'erreur devient plus petite ;
2) pour un même pas, l'approximation est meilleure avec la méthode modifiée d'Euler ;
3) la valeur de p est plus grande pour la méthode modifiée d'Euler que pour la méthode d'Euler classique ; donc, la première converge plus rapidement.

La première idée est une relation générale qui s'applique à toutes les méthodes d'approximation, tandis que les autres idées sont spécifiques aux méthodes explorées et comparées. Si Norman reconnaît que la première idée est assez « évidente », il se fait un devoir et un plaisir de montrer aux étudiants, à travers les deux autres notions, qu'une méthode de calcul peut être améliorée, parfois en utilisant des idées très simples. Après tout, comme nous l'avons déjà vu, l'amélioration des méthodes peut être un défi important pour le chercheur en mathématiques ; rappelons, par exemple, le programme complexe écrit par Norman dont l'algorithme initial était trop lent pour arriver aux résultats cherchés (voir la page 60).

Dans presque tous les exemples décrits par nos participants, le professeur donne beaucoup de soutien aux étudiants dans l'écriture de leurs programmes ; rarement visent-ils un

niveau d'interaction pleinement autonome avec la programmation (N5). Par exemple, même si ce sont les étudiants qui créent les programmes dans l'exemple ci-avant, Norman leur spécifie presque toutes les étapes nécessaires pour le faire. Mais l'encadrement strict du professeur ne sert pas qu'à aider l'étudiant. Nathan, qui enseigne des méthodes numériques dans un cours de mathématiques financières, explique : « Souvent je leur donne tout le "frame" du code […] je veux faire rouler rapidement les programmes quand je corrige. » Alice parle aussi de spécifier des éléments comme les structures d'entrée et de sortie (input/output) pour faciliter son processus d'évaluation. À l'occasion, elle et Alain fournissent même des programmes complètement écrits où les étudiants n'ont qu'à procéder à des modifications (N3) pour faire leurs devoirs.

Programmer ou ne pas programmer, telle est toujours la question

Cela nous ramène à cette question récurrente dans nos analyses : est-il vraiment nécessaire pour les étudiants d'écrire des programmes eux-mêmes ou est-il suffisant pour eux d'explorer des algorithmes en utilisant des programmes dans lesquels ces algorithmes sont déjà implantés ? On pourrait imaginer, par exemple, que les étudiants de Norman, dans l'exemple précédent, puissent faire la même exploration à partir de fonctions préprogrammées, en manipulant l'interface d'un programme qu'on leur aurait fourni (N1) ou en regardant une simple présentation (N0) faite par leur professeur.

Selon plusieurs mathématiciens rencontrés, du point de vue de l'étudiant, la tâche d'implémenter un algorithme peut être très riche. Par exemple, Nathan explique qu'en amenant ses étudiants à programmer des méthodes, il peut mieux voir s'ils les ont vraiment comprises. Il avance que l'acte de transformer une méthode en une suite d'instructions dans un langage de programmation requiert une compréhension profonde de la méthode ; la programmation peut donc aider les étudiants à mieux comprendre les méthodes elles-mêmes. Rappelons la conviction de Paul à l'effet que ses étudiants « apprennent mieux en programmant. » Selon le raisonnement de Nathan, on pourrait se demander si l'écriture des programmes pourrait aider les étudiants à mieux comprendre les concepts et les processus qu'ils apprennent, par leur prise en charge de la transposition informatique de ces objets. C'est exactement l'argument proposé par Leron et Dubinsky (1995) au regard des idées d'algèbre abstraite.

La conception des programmes peut aussi sensibiliser les étudiants en mathématiques à la différence entre la compréhension théorique d'une méthode et son application en pratique. Dans son cours de méthodes numériques, Alice prépare des tâches qui, sans être très complexes sur le plan théorique, sont exigeantes au regard de la programmation. Par exemple, elle demande aux étudiants de créer un programme en *MATLAB* pour calculer et représenter visuellement les bassins d'attraction de Newton pour les racines de la fonction $y = x^3 - 1$. Elle explique : « The central message there is that implementing the code is different from just writing down the algorithm. There are other things you have to worry about: how do you input data, how do you manage data, how do you output data. » Ses étudiants doivent aussi adapter des méthodes apprises pour les utiliser dans de nouveaux contextes. Cet exercice participe au développement de leurs compétences en résolution de problèmes.

Enfin, s'il est presque sûr que les étudiants auront un jour à utiliser des programmes construits par d'autres, la programmation d'algorithmes peut les aider à ouvrir (ou à imaginer l'intérieur) des boîtes, qui resteraient noires autrement. Albert explique :

> Engineers might not agree with me and say it's so useful as part of an engineering program that [students] be proficient in using certain packages. And they can learn a lot in a sense because [...] they could be computing very quickly using a package. But, would they really understand what it means and what is involved? [...] independently programming relatively simple algorithms should be part of a mathematics program. [...] it gives students insight into how packages actually work.

Pour illustrer sa perspective, Albert décrit une tâche qu'il exige de ses étudiants dans son cours d'algèbre linéaire : l'écriture d'un programme qui met en œuvre l'algorithme d'élimination gaussienne. Il existe des millions de logiciels implémentant cet algorithme ; mais Albert tient à l'idée que la programmation de l'algorithme peut aider les étudiants à mieux comprendre de tels logiciels, et par extension, leur permettre de les utiliser plus soigneusement. Il est intéressant de noter que, contrairement à ce qu'Albert pense, le mathématicien de notre étude qui travaille dans une école d'ingénieurs est complètement d'accord avec lui. Dans son cours d'éléments finis, Alain invite ses étudiants à développer leurs propres programmes et à explorer des programmes existants afin de les aider à ouvrir la boîte noire qu'ils utiliseront plus tard.

Utilisation d'outils pour réaliser des calculs ou créer des images (T7)

L'utilisation d'outils de calcul ou de représentation graphique créés par d'autres est explicitement mentionnée par plusieurs des mathématiciens que nous avons rencontrés. De tels outils puissants permettent à l'utilisateur d'épargner temps et énergie qui peuvent être réinvestis ailleurs ; par exemple, dans la résolution d'un problème plus complexe ou plus important. Pierre emploie des routines en *Sage* pour effectuer certains calculs ou pour générer certaines images rapidement dans la préparation de ses cours ; il montre seulement les résultats aux étudiants (N0), simplement pour ne pas avoir à reproduire ces calculs ou images au tableau. De façon similaire, Alain encourage ses étudiants à utiliser le calcul symbolique en *MATLAB* (N4) pour calculer des intégrales quand ils sont en train d'analyser des méthodes numériques : « Je ne veux pas qu'ils perdent du temps à faire des calculs à la main. » Il ne veut pas non plus qu'ils perdent du temps à récrire les fonctions préprogrammées déjà disponibles en *MATLAB*. Omar, pour des raisons semblables, permet à ses étudiants d'employer n'importe quel outil lorsqu'ils font les devoirs dans son cours de géométrie différentielle. Il explique : « Some of my assignments are simply too dreadful to do without *Mathematica*, *Maple*, or something like that. » Mais contrairement à Alain, ce mathématicien laisse le choix des outils à ses étudiants (N5), ce qui pourrait les mener à développer l'habileté de distinguer les moments où un outil informatique serait plus approprié que leur crayon ou vice versa.

Ainsi, personne parmi nos participants ne semble nier le fait que les étudiants profitent de l'appropriation de méthodes de calcul ou de représentation impliquant l'utilisation de logiciels. Même Albert, qui cherche à réduire au minimum le recours aux fonctions préprogrammées dans les exercices de programmation qu'il propose à ses étudiants, croit aussi nécessaire l'exposition des étudiants aux fonctions puissantes dont les algorithmes sont trop complexes pour qu'ils puissent les reproduire. Il remet en question néanmoins la nécessité d'aider les étudiants dans l'apprentissage de tels outils, car la documentation associée est généralement suffisante. Il maintient aussi que les étudiants doivent tous vivre des expériences de programmation, avant ou en même temps qu'ils utilisent des boîtes noires. Albert n'est pas le seul à proposer ces idées : Alice doute aussi de la nécessité de fournir beaucoup de soutien aux étudiants dans leur apprentissage des outils et Alain dit qu'il forcerait tout étudiant en mathématiques à programmer pour mieux comprendre le fonctionnement de logiciels.

Mais certains participants ne sont pas complètement d'accord avec ce raisonnement. Anthony, par exemple, se demande s'il est vraiment nécessaire pour les étudiants d'ouvrir certaines boîtes noires. Dans ses cours, il utilise et encourage l'utilisation par ses étudiants de *Wolfram Alpha,* un outil qui est une sorte d'hybride entre un moteur de recherche et une grande calculatrice. Ce qui différencie l'usage d'un tel outil de l'activité de programmation est que le code qui en contrôle le fonctionnement n'est pas accessible ; l'étudiant travaille uniquement au niveau de l'interface (N1). En fait, dans le cas de *Wolfram Alpha,* quand on entre des commandes, on n'a pas l'impression de travailler avec un programme informatique puisqu'on n'a pas à suivre des règles de syntaxe rigides ; un des aspects qui rendent cet outil très facile à utiliser est qu'on peut y entrer des commandes en employant un langage familier. Anthony justifie ainsi l'utilisation de *Wolfram Alpha* afin de faire des calculs ou de générer des graphiques : « Leave that to the computers. You don't have to know how to program it. You need to know when to use it and how to interpret what you got out. » Anthony nous rappelle, à l'instar de plusieurs autres participants, que les étudiants emploient chaque jour des boîtes noires de tout type ; par exemple, la calculatrice ou un théorème dont la preuve n'est pas comprise sont autant de boîtes noires pour les étudiants qu'un outil comme *Wolfram Alpha*. Selon Anthony, l'essentiel est que les étudiants soient capables d'interpréter et d'appliquer correctement les résultats qu'ils obtiennent, ce qui pourrait être soutenu par une compréhension avancée du matériel théorique. Si les étudiants savent comment programmer, s'ils ont une idée de comment *Wolfram Alpha* fonctionne*,* tant mieux ; mais peut-être n'est-ce pas nécessaire.

Anthony ajoute qu'il est très important de faire connaître aux futurs enseignants les outils puissants qui sont accessibles à leurs étudiants ; des outils comme *Wolfram Alpha* ou *Google* sont disponibles en ligne gratuitement et leur présence peut aider grandement les étudiants dans leur apprentissage et dans la complétion de leurs devoirs (ou même trop si l'enseignant n'est pas conscient de leur existence). D'autres mathématiciens mentionnent la pertinence de savoir utiliser les logiciels statistiques dans ce champ d'étude. Et Alain, qui enseigne à de futurs ingénieurs, profite de son cours de méthodes d'éléments finis pour que ses étudiants apprennent à employer le logiciel commercial *COMSOL.* Contrairement aux idées d'Albert ou d'Alice, Alain passe quelques séances du cours à présenter l'outil et à en expliquer

l'utilisation. L'apprentissage de cette boîte noire est essentiel pour résoudre certains des gros problèmes du cours ; la connaissance du logiciel est par ailleurs un atout supplémentaire face aux futurs employeurs. Alain explique : « Ça va être leur carrière professionnelle. […] Un ingénieur utilise beaucoup de boîtes noires. » Mais si l'utilisation des boîtes noires est inévitable, Alain ajoute que ses étudiants doivent apprendre à s'assurer de la validité des résultats. Il insiste alors sur la vérification du fonctionnement des programmes, avec des bases de données existantes, par exemple. Comme Anthony, il souligne aussi l'importance de la compréhension de la théorie pour aider à l'interprétation et au traitement des résultats. Et finalement, comme Albert, il encourage les étudiants à écrire et à explorer des morceaux de code pour avoir une idée de ce qui se passe à l'intérieur d'un programme comme *COMSOL*.

En conclusion, ce sur quoi tous nos participants s'entendent est que les étudiants, tout comme les mathématiciens eux-mêmes dans leur recherche, doivent savoir profiter des outils qui leur sont disponibles mais, en même temps, ils doivent apprendre à les utiliser avec beaucoup de soin. Alain dit toujours à ses étudiants : « L'intelligence n'est pas dans l'ordinateur, elle est devant l'ordinateur. » Le débat est plutôt sur le degré d'expertise en programmation qui est nécessaire pour l'étudiant en mathématiques. Évidemment, s'il sait seulement comment utiliser des boîtes noires, alors il est limité par les fonctions disponibles ; car tout outil a ses limites. Mais pour ne pas être confiné à l'utilisation de boîtes existantes, il doit être capable de développer ses propres outils, d'instrumentaliser l'ordinateur ; il a donc besoin d'acquérir des compétences de programmation.

4.3.4 Le développement de compétences de programmation

Le cours de base : utilisation, lecture ou écriture des programmes informatiques (T8)

Puisque les étudiants ne sont pas toujours initiés à la programmation avant d'entamer leur baccalauréat, il revient à l'université d'offrir pareille initiation. Tous nos participants sont d'accord : il doit exister un cours pour développer des compétences de base en programmation.

Et dans la plupart des premiers cycles en mathématiques, un tel cours existe.[6] Dans ce cours, l'utilisation, la lecture ou l'écriture des programmes informatiques (T8) deviennent les tâches principales, alors que les tâches « plus mathématiques » occupent un statut secondaire. En effet, plusieurs interviewés soulignent l'importance de ce changement de foyer : de l'utilisation de programmes pour résoudre des problèmes mathématiques, vers un engagement dans l'activité-même de programmation. Barbara, par exemple, suggère que « if you're trying to just use [a computer language] to do stuff that's related to some theory that you're learning, then you don't really learn to program. And then you can't be relied upon to use it. » Alice fait une analogie intéressante avec la façon dont la preuve est souvent enseignée : « For many students, the first introduction they have to a proof is in a calculus class, where they see a limit. But they are thinking about the limit; we are trying to show them proof. There's a mismatch of expectation there. Right? » Sur ce sujet, l'expérience concrète d'Omar dans l'évolution de son cours d'introduction à *Mathematica* est très parlante. Il avait composé dans le cadre de ce cours des problèmes très intéressants sur le plan mathématique ; mais il a fini par les enlever du cours puisqu'ils distrayaient les étudiants occupés à apprendre à programmer.

Les justifications pour le cours de base sont diverses et dépassent l'acquisition simple des compétences de base en programmation. Quelques participants suggèrent en effet que la programmation aident au développement mathématique global de l'étudiant. Albert réfléchit :

> What did I learn from programming? Organizing your thoughts, thinking in terms of algorithms, thinking in terms of efficiency [...] Thinking also in terms of simplicity, right? [...] I think students learn a lot from programming.

Alain, comme Albert, suggère que l'apprentissage de la programmation aide à penser d'une façon logique et à travailler avec l'abstraction, deux habiletés qui sont utiles dans la résolution de problèmes et dans la rédaction de preuves formelles. D'autres mathématiciens réfèrent aux emplois futurs des étudiants. Ainsi, Olivier conclut que savoir programmer « est quelque chose qui peut être pratique dans leur vie professionnelle. »

---

[6] Un seul des premiers cycles en mathématiques discutés en entrevue ne contient pas de cours de base. Barbara déplore cette absence qu'elle voit comme un frein énorme à l'intégration de la programmation dans ses cours.

Si nos participants s'entendent tous sur l'importance d'un premier cours, leurs opinions divergent quant au contenu précis de ce cours et à la manière dont les étudiants devraient y « apprendre à programmer ». Nous présentons ci-après les grandes questions qui font débat.

Apprendre un langage et/ou comprendre le concept d'algorithme ?

Typiquement, le cours de base met l'accent sur l'apprentissage d'un langage spécifique à travers l'écriture de plusieurs programmes dans ce langage. Étant donné la grande diversité des langages, la première question abordée par nos participants est souvent : comment choisir le langage à enseigner ?

Certains mathématiciens accordent un poids important à ce choix. Pierre explique : « Si l'on change le langage de programmation, on peut en fait changer la façon dont on approche un problème. Et donc, c'est pour ça que je pense que c'est une bonne idée d'apprendre plusieurs langages de programmation. » Quelques participants, comme Omar, favorisent un outil comme *MATLAB*, *Maple* ou *Mathematica*, dans lequel les étudiants peuvent écrire des programmes mais aussi avoir accès aux fonctions préprogrammées puissantes qui ont été construites spécifiquement pour les mathématiciens. D'autres participants, comme Alain, préfèrent les « vrais » langages de programmation ; par exemple, des langages comme *C++*, qui sont fondamentaux en informatique. La fréquence d'utilisation au sein d'un département, la facilité d'apprentissage et le prix des licences d'exploitation des environnements associés pèsent également dans la décision. Ben, par exemple, explique que son département a fait le choix du langage *VB.Net*, car il est très facile à apprendre. Il dit que « it's a quick way to introduce students to programming so they don't get frustrated. It's a very forgiving language. For the real world, they need to do more *C++*. Something like *C++*, or *Java*. A more modern language that employers would want. »

À l'opposé, le choix du langage lui-même est moins important pour certains, car les objectifs de ce cours de base doivent être beaucoup plus larges. Olivier contre l'argument de Pierre rapporté ci-haut : « Je ne pense pas que ça vaille la peine d'enseigner plusieurs langages de programmation. [...] à la fin, les principes sont les mêmes. [...] si les étudiants ont une base, c'est très facile d'apprendre un autre [langage] s'ils veulent le faire. » Olivier suggère alors de commencer avec l'idée générale d'algorithme, en dehors du contexte d'un langage spécifique.

Alice est critique des cours de base existants parce qu'ils n'accordent pas suffisamment d'importance au rôle crucial des algorithmes. Elle explique : « The very first exposure that students have to computer programming right now is in computer science courses, in which they learn one language and the focus is the language and the syntax of the language. I think that's problematic because they don't understand algorithms at an algorithmic level. » Même le cours d'Omar, qui relève du département de mathématiques, n'arrive pas explicitement au concept d'algorithme, le situant en dehors des buts visés par le cours. Évidemment, les étudiants dans ce cours peuvent développer implicitement une certaine compréhension des algorithmes, puisqu'ils écrivent plusieurs programmes. Le problème, selon des mathématiciens comme Alice, est que cela ne suffit peut-être pas pour préparer les étudiants à travailler d'une façon qui ne dépende pas d'un langage spécifique et qui ait la flexibilité requise pour s'adapter aux différents environnements de travail. Adèle ajoute que « les langages évoluent et on veut pouvoir s'adapter à ce qui a évolué. » Ces mathématiciens proposent donc que les idées plus générales concernant le développement des algorithmes, que plusieurs considèrent comme provenant du domaine informatique, mais qui rejoignent les mathématiques par leur visée de généralité, reçoivent beaucoup plus d'attention dans un cours de base en programmation.

Qui doit être responsable de ce premier cours ?

Un autre débat concerne le département qui devrait être responsable de ce premier contact avec la programmation. Quelques mathématiciens insistent pour que ce soient les informaticiens ; comme l'affirme Barbara, « they're the experts. » C'est ce qui se passe dans plusieurs baccalauréats en mathématiques actuels. Nathan, évoquant sa propre formation, souligne un problème avec ce choix : « Même si j'ai fait un peu de programmation, ça n'a jamais été bien mélangé avec mes mathématiques. » Un cours enseigné par un département de mathématiques ferait-il mieux ce lien entre la programmation et les mathématiques ?

Même si c'était le cas, il faudrait encore trouver un mathématicien prêt à enseigner le cours de base, ce qui pourrait se révéler difficile. Après tout, ce n'est pas parce qu'un mathématicien sait programmer, et même à un niveau expert, qu'il est prêt à enseigner un cours de base en programmation. Ben tient un discours très honnête sur la question : « I get tired of the details of coding. [...] I have, after all, been coding for many years [...] And so,

teaching loops is not my most favourite thing. [...] it would be like teaching how to play C-major scale on a piano. It's important, it's valuable, it's fantastic. I just don't want to teach it. » De même, Omar, au moment de créer son cours d'introduction à *Mathematica,* a insisté pour ne pas être condamné à l'enseignement de ce cours ; sa solution fut de développer des notes de cours que toute personne compétente puisse utiliser pour l'enseigner.

Une formation de base ... et puis après ?

Nous avons déjà vu que les autres enjeux (E1, E2 et E3) peuvent être servis par l'engagement des étudiants dans des activités impliquant la programmation. La prise en compte de ce quatrième enjeu peut alors s'avérer très utile à l'atteinte des autres. En effet, il y a une relation réciproque entre l'enjeu E4 et l'intégration de la programmation dans des activités mathématiques : pour être capable d'écrire ses propres programmes (N5), l'étudiant doit avoir certaines compétences de programmation et c'est en écrivant des programmes dans divers contextes que ces compétences de programmation se développent. À l'inverse, si les étudiants sont perçus comme ayant peu de compétences de programmation, un professeur pourrait décider de ne pas intégrer la programmation dans ses cours, les étudiants ne peuvent donc pas développer des compétences de programmation, et ainsi de suite.

Plusieurs des mathématiciens interrogés se préoccupent de ce qu'il advient après le premier cours en programmation. Une idée semble partagée par tous : les étudiants doivent avoir l'occasion d'ajouter à leurs compétences de base. Comme l'affirme Alain : « C'est dans la durée que [la programmation] devient utile. Si on ne fait qu'un seul cours, ça sert à rien. » Mais, doit-on rendre obligatoire la poursuite du développement de ces compétences ? Selon quelques mathématiciens, il n'est pas nécessaire de l'exiger de chaque étudiant au-delà du cours de base. Olivier explique : « Ça peut être son choix. [...] je pense qu'il est important que [l'étudiant] ait une base qui lui permette d'aller plus loin s'il le veut. » D'autres mathématiciens sont un peu plus exigeants. Par exemple, Paul s'exclame : « Let's standardize [a programming language] so that they learn something and then they can use it, and then they do use it [...] Then it will become second nature, it'll just be, you know, like using a calculator. »

Pour que les compétences de programmation s'intègrent à leur boîte à outils, les étudiants doivent avoir plusieurs occasions, tout au long du baccalauréat, d'écrire leurs propres

programmes informatiques dans différents contextes mathématiques. Les sections précédentes ont donné des exemples de telles opportunités d'interagir avec la programmation. Mais il y a un sentiment partagé par presque tous les mathématiciens rencontrés qu'on pourrait, et même qu'on devrait, faire davantage. Nathan suggère : « La place de la programmation dans un bac en mathématiques devrait être *énormément* plus grande que ce que moi j'ai vécu. »

Des avenues sont proposées. Rappelons les suggestions des participants d'étoffer les cours d'équations différentielles et d'algèbre linéaire par des activités requérant la programmation. Le partage des ressources existantes en est une autre. Paul, par exemple, rend disponibles aux professeurs dans son département toutes les ressources qu'il utilise pour son intégration unique de la programmation dans un cours de probabilités. Quelques mathématiciens suggèrent même l'ajout de nouveaux cours. Alain fait valoir le potentiel d'un cours bâti sur les notions venant de la physique pour initier les étudiants aux « nouvelles » méthodes de calcul ; Ben propose la création d'une série de cours de modélisation interdisciplinaire ; Pierre encourage l'inclusion d'un nouveau cours de mathématiques expérimentales dans lequel « on va faire des expériences, [...] on va essayer de faire des observations, [...] des conjectures, essayer de démontrer ce qu'on a, [...] montrer que les mathématiques peuvent être une science expérimentale. » On remarque ainsi l'intérêt des mathématiciens pour engager leurs étudiants dans des activités qui ressemblent à leur propre travail de recherche. Cependant, l'intégration de plusieurs de ces activités de programmation pourrait exiger des changements importants à un baccalauréat en mathématiques, changements qui nécessiteraient beaucoup plus que la conviction d'un mathématicien isolé.

Le problème de l'évaluation : rédaction de devoirs et passation d'examens (T9)

Même si plus d'activités de programmation sont intégrées dans un baccalauréat en mathématiques, leur présence dans les évaluations des cours déterminera leur importance et, par extension, l'importance des compétences de programmation. Alors, comment les étudiants de nos participants sont-ils évalués sur leur programmation ?

Les mathématiciens que nous avons rencontrés semblent tous ouverts à l'intégration d'activités de programmation dans les devoirs ou dans un environnement de laboratoire. Parfois, la programmation n'est pas vraiment évaluée et se voit alors attribuée un statut

« extrascolaire ». D'autres fois, les étudiants ont à remettre leur travail. Les différentes méthodes d'évaluation favorisées par nos participants reflètent les méthodes de communication employées dans le contexte de recherche. Quelques professeurs corrigent la programmation de leurs étudiants principalement en termes des résultats obtenus. Albert explique ce choix : « The most important thing is that they get the correct results. » Norman ajoute : « I can tell by looking at the answers that they got whether they've done it correctly or not. » Dans certains cas, la découverte d'un résultat bizarre pourrait entraîner le professeur à rouler le programme sur plus d'exemples ou à regarder rapidement le code ; sinon, on suppose une qualité suffisante de l'algorithme et du codage. À l'opposé, d'autres participants tiennent à examiner le code, ce qu'Omar justifie en disant : « When they go to the computer, they need to prove to me that they know what they're doing. » La méthode d'évaluation d'Alice est structurée pour prendre en considération tout niveau de la programmation faite par ses étudiants :

> They would upload their code, I would run a script with randomized inputs, and I would check their outputs against what I expected [them] to be. If their codes failed to compile, or if they didn't give the answer expected, they get a zero. If the code worked, I would look at it. If they didn't document to the ratio of about one line of documentation per line of code, again a zero. [...] if there was sufficient documentation, then I would actually look at the code. And then I would assess it on flexibility and quality of documentation.

Si cette approche semble être stricte, Alice la justifie par le fait que les programmes à réaliser au niveau baccalauréat ne sont pas très complexes. En fait, d'autres mathématiciens le constatent et n'évaluent pas par conséquent des choses comme l'efficacité du programme. Nathan l'explique : « Ce ne sont pas des programmeurs que j'évalue, ce sont des gens qui apprennent des mathématiques. » Les étudiants, par conséquent, n'apprennent pas à voir la programmation comme une composante importante du travail des mathématiciens.

Au regard de l'intégration de la programmation dans un examen « traditionnel », quelques participants ne sont pas très enthousiastes. Norman suggère tout simplement que cela n'est pas possible. Alain l'a essayée, mais il doute qu'il la refera en raison de l'expérience qu'il a vécue :

> Il y avait des étudiants qui ne savaient pas fonctionner, et ce n'étaient pas leur faute. [...] ça fonctionnait pour moi, mais je savais par où passer. Et, bon, les étudiants, souvent ils allument des choses, ils éteignent d'autres choses et ça ne marche plus alors. Donc c'est délicat.

Paul évite complètement ces situations en intégrant des problèmes qui demandent aux étudiants d'écrire et d'interpréter du code écrit. Après tout, ce n'est pas la syntaxe qui l'intéresse ; il évalue ses étudiants sur la logique de leurs programmes. Il attribue ainsi un poids supérieur au développement d'algorithmes qu'aux autres compétences évaluées en examen.

Enfin, même si une compétence est évaluée, sa valeur perçue est souvent proportionnelle à son impact sur la note finale d'un cours. C'est encore Paul qui nous mène à une réflexion importante. Les activités de programmation de ses étudiants peuvent demander plus de temps que toute autre activité du cours ; mais elles valent moins. Le mathématicien explique ainsi cette apparente contradiction :

> They can just ignore this part of the course and still get eighty-five percent if they do everything else right. […] But almost none of them ignore it. Because they like it. They really feel that they learn from it and they write comments in the evaluations like "I always did the [programming] problem first!"

Si les étudiants semblent vraiment aimer cet aspect du cours et en apprendre beaucoup, pourquoi ne pas y accorder plus d'importance ? Paul explique que l'appropriation de la théorie probabiliste (E1) est encore l'aspect le plus important dans son cours ; la programmation est simplement un outil pour y arriver.

5 Comparaisons et justifications institutionnelles

Ce chapitre aborde nos dernières questions de recherche. À partir de la description des pratiques présentée au Chapitre 4, nous comparons l'intégration de la programmation dans les pratiques en recherche de nos participants et dans les pratiques en apprentissage de leurs étudiants. Un résumé des similarités et des différences marquantes est d'abord présenté. En reprenant ces résultats sous l'angle des contextes institutionnels, nous proposons ensuite quelques pistes d'explication.

5.1 Similarités prometteuses, différences intrigantes

La description que nous avons faite des pratiques a mis en évidence des moments précis où l'on peut noter une ressemblance frappante entre ce que les mathématiciens font avec la programmation en recherche et les expériences qu'ils planifient pour leurs étudiants. Dans la section 4.3.1, nous avons vu que les activités planifiées par quelques mathématiciens transforment la simple réception de résultats mathématiques en un cycle d'exploration semblable à celui vécu par des mathématiciens purs : les étudiants sont conduits à faire des observations, formuler des conjectures, les vérifier, créer de nouvelles conjectures, etc. La section 4.3.2 révèle quelques exemples où les étudiants utilisent des programmes pour cerner les régularités qui conduisent aux modèles mathématiques ou pour expérimenter avec les modèles, tout comme les mathématiciens appliqués le font en recherche. Et finalement, quelques exemples à la section 4.3.3 reflètent le travail d'un mathématicien qui cherche à développer de nouveaux outils informatiques.

Ceci étant dit, la discussion présentée au Chapitre 4 fait aussi ressortir des différences intrigantes entre les pratiques des participants et celles de leurs étudiants. La section 4.3.4 en soi indique une différence évidente entre les deux contextes : les étudiants sont vus toujours comme étant en mode d'apprentissage au regard de la programmation, alors que les mathématiciens ont déjà terminé cette première appropriation de la programmation. Cet écart mène à des différences évidentes entre les pratiques ; par exemple, la complexité de la programmation en recherche est en général beaucoup plus grande que celle en apprentissage. Il est aussi rare qu'un étudiant doive décider par lui-même de l'ensemble du processus de sa

programmation, y compris le fait d'y recourir ou non (le seul exemple, celui d'Omar, est rapporté à la section 4.3.3) ; comme on peut s'y attendre dans un contexte d'apprentissage, les activités et marches à suivre sont presque toujours bien délimitées par le professeur. En fait, s'il existe d'autres différences générales intéressantes (les étudiants ne développent pas de nouvelles théories, des modèles ou des outils, ne choisissent pas leurs problèmes, ne valident pas toujours leurs explorations, ne sont pas mis en contact avec des preuves par ordinateur, etc.), les différences qui semblent être soulignées le plus par nos participants concernent les niveaux d'interaction avec la programmation. Dans plusieurs cas, les mathématiciens sont beaucoup plus actifs dans l'activité de programmation que leurs étudiants. Mais pour trois de nos participants, c'est plutôt l'inverse : même si ces mathématiciens ne programment pas dans leur recherche, ils intègrent la programmation dans leurs cours.

Quand il existe des différences marquées entre la programmation d'un mathématicien comme chercheur et celle de ses étudiants, le mathématicien les explique souvent par le poids des institutions dans lesquelles il évolue. De façon similaire, aucun participant ne décrit la programmation idéale d'un étudiant de premier cycle sans mentionner certaines contraintes imposées par les contextes institutionnels, des contraintes à surmonter ou des contraintes à respecter. Il nous apparaît donc opportun de discuter dans la section qui suit des comparaisons entre les pratiques en recherche et celles en apprentissage dans les contextes respectifs du curriculum, du département, de la communauté mathématique et de la société.

5.2 L'impact des institutions : du curriculum à la société

5.2.1 Le curriculum

Le curriculum d'un baccalauréat en mathématiques présente plusieurs éléments intimement liés qui influencent la décision des professeurs d'inclure des activités de programmation dans un cours donné. Les objectifs du cours, le temps qu'on a pour les accomplir et les prérequis qui forment des liens entre des cours sont tous mentionnés par nos participants comme des contraintes importantes.

Les objectifs

Selon les mathématiciens rencontrés dans notre étude, les objectifs d'un cours déterminent largement la place de la programmation : en général, plus les objectifs sont purs ou abstraits, moins les étudiants sont invités à interagir avec des programmes informatiques.

Quand Norman résume ses propres activités d'intégration de la programmation, il dit : « I guess most of the times the programming has involved teaching something that is very much programming related. » Dans sa recherche, Norman écrit des programmes informatiques pour résoudre des problèmes variés, y compris des problèmes plus purs ; par contre, ce n'est que dans les cours appliqués qu'il planifie des activités de programmation pour ses étudiants, lesquelles sont toutes liées à la modélisation. Tous nos participants considèrent que les objectifs d'un cours appliqué se prêtent naturellement à l'intégration de la programmation. Il n'est donc pas surprenant qu'il n'y ait, typiquement, qu'une petite différence entre le niveau de programmation des mathématiciens appliqués (N5) et celui où ils convient leurs étudiants (N4). Philippe, en réfléchissant aux cours où les étudiants devraient écrire leurs propres programmes informatiques, explique que ce sont « surtout les cours de maths appliquées. Calcul scientifique, analyse numérique, évidemment ces cours-là. Donc ça c'est vraiment des cours où l'idée est de développer des méthodes, puis être capable [...] de les appliquer. [...] si je les enseignais, évidemment je ferais des choses comme ça. » Les objectifs des cours appliqués semblent classer la programmation parmi au moins *les pratiques admises* dans ces cours.

Mais pour les autres cours? Selon nos participants, les objectifs des cours plus théoriques et abstraits sont plus difficilement compatibles avec l'intégration d'activités de programmation. Les mathématiciens appliqués suggèrent même que l'utilisation de l'ordinateur n'est pas appropriée dans ces cours, la reléguant ainsi parmi *les pratiques contestées*. Alain, par exemple, explicite sa vision avec beaucoup de confiance :

> En théorie des groupes je dirais non.
> *Non?*
> Non.
> *Pas du tout?*
> Pas du tout.

Comme Artigue (2002) le prédit, certains mathématiciens appliqués ont même du mal à expliquer pourquoi ils n'envisageraient pas d'intégrer des activités de programmation dans un cours comme celui de la théorie des groupes, tant la décision leur semble être évidente. D'autres, au moment d'être confrontés avec la possibilité d'intégrer la programmation dans ce type de cours, suggèrent que cela pourrait détourner l'attention des étudiants de ce qu'ils doivent vraiment y accomplir. Albert, par exemple, explique : « Some people I think have suggested we should use computers in [Discrete Mathematics]; I am absolutely against that [...] computers can be used everywhere, but also students need to learn logical reasoning, also to think why something is true, think about that in their head, be able to prove it. » Des cours d'algèbre abstraite, de topologie ou d'analyse, où la norme est d'exposer les étudiants aux mathématiques formelles par une présentation magistrale, ne sont donc pas, aux yeux de nos mathématiciens appliqués, des contextes naturels pour inclure des activités de programmation.

Si les mathématiciens purs paraissent plus ouverts à la possibilité d'ajouter des activités de programmation dans certains cours de mathématiques pures, ils remettent en question néanmoins l'importance de telles activités. Ces mathématiciens maintiennent, tout comme leurs collègues appliqués, que l'objectif principal d'un cours théorique demeure celui de mieux comprendre la théorie. Olivier précise cette position :

> Il faut toujours garder un équilibre. [...] le cours de géométrie différentielle, on ne peut pas le transformer en juste un cours de calcul, où on prend les formules, on a un programme, on calcule avec les programmes les courbures... [...] le sens du cours est les concepts mathématiques [...] Donc [l'ordinateur] peut jouer un rôle plutôt auxiliaire.

Pierre croit possible l'intégration d'activités de programmation dans les cours plus abstraits (p.e., celui de topologie) ; mais tout comme Olivier, il voit ces activités comme un ajout dans la mesure où elles sont pensées pour aider les étudiants à mieux comprendre certaines notions. Pour ne pas transformer ses cours de probabilités en « cours de calcul », Philippe se sert de programmes qu'il a écrits pour expliquer des idées aux étudiants, mais n'exige pas d'eux qu'ils programment à leur tour. Même s'il ne programme pas dans sa recherche, ce mathématicien s'applique à programmer dans son enseignement, car il croit au pouvoir de l'illustration de la théorie que lui permet l'ordinateur. Toutefois, il pense que les cours en mathématiques pures ne devraient pas exiger des étudiants qu'ils programment, tout comme ils ne devraient pas non plus avoir comme objectif le développement de compétences de programmation.

Il est intéressant de noter ici que, malgré leur ouverture à l'intégration de la programmation, les mathématiciens purs sont les participants chez qui nous observons la plus grande différence entre leur propre programmation et celle de leurs étudiants. Si Olivier nous a fourni l'exemple emblématique pour illustrer des pratiques en mathématiques pures impliquant la programmation (voir 4.2.2), ce mathématicien n'écrit aucun programme pour son enseignement et n'encourage pas davantage la programmation par ses étudiants. De façon similaire, en dépit de ses propres illustrations du contenu enseigné qu'il obtient de programmes informatiques, Pierre ne donne aucune occasion à ses étudiants de prendre charge l'activité de programmation dans ses cours ; il est cependant le mathématicien pur qui programme presque autant que les mathématiciens appliqués. Et Omar, qui développe des programmes complexes aussi facilement que Pierre, laisse à ses étudiants l'initiative de grimper vers de plus hauts niveaux de programmation, avec pour effet que les étudiants se cantonnent souvent au niveau N0. Évidemment, être capable d'ajouter des activités de programmation dans un cours ne suffit pas à justifier leur intégration ; il y a d'autres aspects à considérer.

Les contraintes de temps

Omar évoque souvent les contraintes de temps inhérentes à l'enseignement. En regardant une illustration qu'il a créée pour ses étudiants à l'aide d'un de ses programmes, il mentionne ainsi : « Il y a tellement de matière dans le cours qu'il me semble que ce serait exagéré de leur demander aussi de programmer ça. Mais, si on avait le temps, encore davantage, bien ça serait chouette, mais il y a toujours... il y a une limite. » Omar parle ici du temps de l'étudiant, qui a déjà à gérer un curriculum chargé. À un autre moment pendant l'entrevue, il mentionne aussi le temps du professeur, qui planifie les activités d'apprentissage. Paul dit que la conception d'un seul problème impliquant la programmation peut lui demander plusieurs jours. « And that's expensive, » il explique, « whereas like a typical other problem [...], an ordinary problem on a homework assignment is something that, you know, maybe an hour you can come up with a really good problem. » Préparer n'importe quel cours requiert un certain travail du professeur ; mais si on ajoute à cette préparation la création de nouvelles activités (intégrant la programmation ou pas), cela peut augmenter considérablement le temps que le professeur doit consentir. C'est en quelque sorte le prix associé à *l'innovation* et à la transformation de *pratiques contestées en pratiques admises.*

Notons que l'exigence de temps pour innover peut être beaucoup plus lourde pour ceux qui ont des compétences de programmation limitées. Nathan, par exemple, est un mathématicien qui n'utilise presque jamais l'ordinateur dans sa propre recherche ; rappelons qu'il nous a fait part d'un seul moment où il a écrit un programme pour vérifier une conjecture avant de s'attaquer à une preuve formelle (voir la page 64). Ce professeur intègre quand même des activités appliquées de programmation dans les cours qui l'exigent par leurs objectifs ; il a développé aussi, de sa propre initiative, des programmes pour illustrer la théorie abstraite qu'il enseigne. S'il a clairement surmonté son manque initial d'expérience avec l'activité de programmation, il ne s'en satisfait pas pour autant : « J'ai l'impression que je dois en savoir plus [...] parfois c'est un peu prenant de préparer les trucs avec la programmation si l'on n'en fait pas tous les jours de façon constante. »

Sans doute parce que la programmation fait davantage partie intégrante de leur travail, les mathématiciens appliqués l'envisagent moins comme une surcharge de temps qui leur pose problème. Cela ne signifie pas qu'ils ne soient pas conscients des contraintes importantes qu'elle peut imposer aux collègues. En parlant des professeurs qui choisissent de ne pas intégrer la programmation dans leurs cours, Barbara explique : « It is very time consuming to put a computer component into a course. It does require a lot of work on the instructor's side, compared to just teaching the usual way, without computers. »

Les prérequis

Les prérequis qui déterminent quels cours sont préalables à un cours donné au sein d'un programme constituent un autre élément mentionné par nos mathématiciens purs. Pour des cours de mathématiques pures, dont les objectifs sont loin d'inclure le développement de compétences de programmation, les prérequis n'incluent pas de cours de programmation. Pierre, en remarquant que le cours de base en programmation ne se trouve pas parmi les prérequis pour les autres cours dans un programme en mathématiques pures conclut : « Il n'est pas nécessaire d'apprendre à programmer dans le programme à l'instant. » Mais dans ce programme on trouve aussi des cours considérés par tous nos participants comme étant des lieux naturels pour l'inclusion d'activités de programmation. Au regard du cours d'analyse de Fourier, Olivier dit : « Il est possible de rajouter l'utilisation des ordinateurs au niveau des

exercices en principe. [...] mais, le problème est que ce n'est pas un préalable. Donc je ne peux pas demander aux étudiants de le faire parce que certains peuvent me dire qu'ils ne savent pas comment le faire. » Barbara est une autre mathématicienne qui justifie la grande différence entre sa propre interaction avec la programmation au niveau N5 et l'interaction typique de ses étudiants au niveau N1 à cause de l'absence de cours préalables en programmation, et ce, dans tous les cours qu'elle donne. Elle explique qu'à cause de l'absence d'un cours de base, ses étudiants sont effrayés à l'idée de programmer au début de ses cours.

Enfin, en raison du curriculum, les activités de programmation qui ressemblent aux pratiques des mathématiciens appliqués sont *admises* dans des cours très spécifiques (p.e., analyse numérique, modélisation), mais ne sont pas nécessairement *partagées* par tous. Par contre, des expériences de programmation qui s'approcheraient de ce que font les mathématiciens purs ne trouvent pas de cours où elles pourraient être incluses naturellement ; elles sont mêmes *contestées* dans certains cours théoriques et abstraits. C'est possiblement pourquoi nos participants suggèrent des changements curriculaires tels que l'obligation de suivre des cours de calcul numérique chaque année ou le développement de nouveaux cours en mathématiques expérimentales, de façon à rendre toutes les pratiques informatiques officiellement acceptées. Mais évidemment, pour procéder à de tels changements, une reconnaissance institutionnelle plus large s'avère nécessaire.

5.2.2 Le département

Le curriculum d'un baccalauréat en mathématiques est créé par un département qui y impose lui-même des contraintes. En effet, certains de nos participants ont commencé par justifier leurs actions en citant des éléments curriculaires, pour révéler plus tard des contraintes plus profondes. Le cas de Paul, décrit ci-après, en est particulièrement éclairant.

Du curriculum au département

Dans sa recherche, Paul participe à des projets qui impliquent la construction et l'utilisation de programmes informatiques complexes. S'il aide à développer et à valider les algorithmes correspondants, il n'écrit aucune ligne de code lui-même ; il interagit avec l'activité de programmation au niveau de l'interface (N1). Dans son enseignement, Paul est le

seul mathématicien pur parmi nos participants qui prépare des activités d'apprentissage où les étudiants doivent écrire leurs propres programmes informatiques (N4) afin de mieux comprendre certaines théories mathématiques abstraites. Pour lui, comme pour Nathan et Philippe, les niveaux d'interaction avec la programmation de ses étudiants sont souvent plus élevés que les siens. Ceci étant dit, le cours de probabilités où Paul intègre la programmation est destiné seulement aux étudiants en sciences ; dans les cours de probabilités suivis dans le programme en mathématiques pures, le professeur n'emploie aucune activité informatique.

Pendant l'entrevue, Paul suggère que le potentiel de programmer pourrait être le même pour tout type d'étudiant. Il dit de l'inclusion d'activités de programmation dans les cours de mathématiques pures : « I think it's the right way to go actually. I think that we're missing an opportunity here. » Alors pourquoi ne pas le faire? Les premières raisons données par Paul concernent les objectifs, la limite de temps et les prérequis du cours, c'est-à-dire les éléments curriculaires discutés à la sous-section précédente. Il dit que les étudiants en mathématiques pures n'auraient pas les connaissances nécessaires pour créer des programmes et, enfin, il faudrait consacrer beaucoup de temps pour déterminer comment intégrer de nouvelles activités dans un cours dont le contenu et la façon de le donner sont déjà bien définis. Notons que Paul a déjà résolu ces problèmes : le cours de probabilités destiné aux scientifiques a commencé de la même façon, sans prérequis de programmation ni activités de programmation. Mais en poursuivant la discussion, il aborde d'autres contraintes plus large : « Academia is a very conservative place. And there's a huge amount of inertia. And there's also a huge amount of independence among the different instructors; and what I'm proposing is something that would require a degree of coordination that I see it difficult to maintain. » Il dit de plus : « I don't hear a lot of people talking about this being a great idea or something like that. »

Les personnalités mathématiques et le degré de coordination nécessaire

Les normes d'un département, qui reflètent en partie le groupe de professeurs qui y travaillent, peuvent d'instaurer en obstacle au regard de l'inclusion d'activités de programmation. D'un côté, la liberté académique est considérée par la plupart des participants comme un principe à respecter sans condition. Pierre dit ouvertement du recours à la programmation dans certains cours : « Je ne veux pas forcer mes collègues à l'utiliser ici ou à

ne pas l'utiliser. » Selon plusieurs mathématiciens, la liberté académique permet aux professeurs d'enseigner avec l'approche qu'ils connaissent mieux, ce qui a pour effet de rendre leurs cours plus efficaces. De plus, cette liberté permet aux professeurs de montrer aux étudiants leur propre personnalité mathématique. Omar explique :

> Il y a des gens qui aiment seulement les résultats les plus généraux, d'autres aiment voir tous les petits contrexemples ou pourquoi ce résultat-là, quand on change un petit peu les hypothèses, c'est faux. Et, je pense que c'est mieux [...] que les étudiants soient exposés à plusieurs personnalités mathématiques et que je puisse enseigner seulement les méthodes et les outils que je maîtrise le plus.

Omar, qui maîtrise l'utilisation de l'ordinateur, peut très facilement l'intégrer dans son enseignement. Ses collègues, par contre, n'ont pas tous les compétences de programmation qui leur permettraient de générer pareilles illustrations ou activités aussi naturellement que lui ; ils enseignent simplement d'une façon différente. Ce mathématicien voit néanmoins le risque de privilégier certaines pratiques au détriment d'autres : « Je me rends compte que mon département est très abstrait [...] Et les étudiants qui sont en mathématiques pures adorent ça. Mais je crois que cela castre certaines de leurs habiletés qui sont absolument essentielles si l'on veut devenir chercheur. »

Si la liberté académique permet aux mathématiciens de faire valoir auprès de leurs étudiants leur propre manière de faire des mathématiques, elle peut aussi mener les étudiants à n'être exposés qu'à un ensemble restreint de personnalités mathématiques. Nathan, un mathématicien plus pur, craint un biais en faveur des mathématiques traditionnelles dans un département, comme c'est le cas dans le département d'Omar. Il dit qu'avec des « collègues qui ont des formations en mathématiques pures, qui ont toujours fait les choses de la même façon, puis qui sont, disons, quinze à penser pareillement [...] il est alors difficile de changer leurs façons de faire, leur façon d'être. Cela est d'autant plus vrai que ce sont tous des gens qui ont réussi leur carrière en étant seulement théoriques. » Nathan suggère que, dans un tel département, la programmation pourrait être *contestée* et n'exister que de façon marginale. Mais l'uniformité du corps professoral d'un département peut également jouer dans l'autre direction : il peut aussi exister un biais si la plupart des professeurs favorisent des méthodes impliquant la programmation. Alice, une mathématicienne appliquée, met en garde contre cette situation : « [If] [e]very professor is trying to make their class more engaging and more

accessible and more applicable [...] everybody's doing very interesting cool stuff with the codes [...] but nobody's proving things. And that becomes a problem. It's as problematic as the other way around. » Le *partage* des pratiques impliquant la programmation ne devrait pas rendre *superflues* ou même *contestées* celles qui ne l'impliquent pas. Ainsi, la solution idéale pour ces mathématiciens semble se trouver dans un équilibre où les étudiants sont exposés aux diverses personnalités mathématiques et aux différentes façons de faire des mathématiques. Mais comment atteindre un tel équilibre ?

Il est important de noter que ceux qui cherchent pareil équilibre ne veulent pas forcer leurs collègues à enseigner d'une façon contraire à leur personnalité. Pour certains participants, coordonner l'intégration de la programmation signifie s'engager dans la recherche d'un consensus départemental qui soit profitable pour les étudiants. Ben explique :

> None of these things are simple. There's a lot of inertia in Universities. [...] You don't just introduce something and it happens. [...] You introduce it one year, and everybody talks about it, and it's a no. And then there's lots of conversations about it [...] because you want people to have something that they truly need, and that has to evolve through discussion.

Si la nature conservatrice des départements universitaires est souvent présentée de façon négative, Ben souligne l'importance de protocoles rigides dans l'évolution des curricula : ils font en sorte que de longues discussions approfondies puissent avoir lieu et favoriser ainsi le développement de solutions riches qui fonctionnent pour tout le monde, y compris les professeurs qui doivent les implanter.

Les ressources humaines

Évidemment, les discussions départementales devraient tenir compte de la faisabilité des changements suggérés : a-t-on les moyens nécessaires pour les faire ?

Tout d'abord, on a besoin de professeurs (ou de chargés de cours) qui soient prêts à donner les cours affectés par les changements. Les trajectoires de nos participants indiquent que le niveau d'expertise en programmation qu'a un professeur n'est pas nécessairement un bon indice de son intérêt à prendre en charge un cours qui implique un nouvel aspect informatique. D'une part, nous avons rencontré des mathématiciens comme Omar et Ben, qui programment souvent dans leur recherche, mais qui préfèrent ne pas enseigner un cours de base en

programmation. D'autre part, nous avons aussi croisé Paul, Philippe et Nathan, qui ne programment presque jamais dans leur recherche, et qui ont tous choisi, avec beaucoup d'enthousiasme, de s'investir dans la réalisation de nouvelles illustrations informatiques ou dans la conception d'activités de programmation pour leurs étudiants. Paul et Nathan expliquent que l'aide d'étudiants aux cycles supérieurs dans le développement des activités a été indispensable. Paul dit même qu'il serait ouvert à intégrer la programmation dans d'autres cours s'il avait le même type de soutien.

Mais les contraintes imposées par les ressources humaines ont des effets qui débordent du développement d'activités de programmation. Alice observe :

> There's a resource issue which is universal in Canadian Universities I think [...] we're pushed to enroll a lot of students. And we don't have the resources to actually do a very diligent one-on-one job with them. [...] if you have a very large [...] class [...] there's no way that you can grade code. Right?

Alice est une mathématicienne appliquée qui a écrit beaucoup de programmes informatiques complexes à travers sa carrière. Si elle préfère inviter ses étudiants à créer des programmes par eux-mêmes (N4), elle leur fournit parfois des programmes déjà développés (N1-N3) car elle ne dispose pas de ressources en nombre suffisant pour évaluer leur code en détail. Elle justifie son approche ainsi : « If it's not assessed in detail, the requirement is shallow, shall we say. I could require you to run a marathon, but if I only checked the fact that you started off, I'm not really checking it in detail; so the requirement is shallow. » Norman affirme que l'évaluation d'activités de programmation dans certains cours à son université ne serait tout simplement pas possible en raison du très grand nombre d'étudiants et du fait qu'on se contente alors des examens comme unique forme d'évaluation. Comme nous l'avons déjà mentionné à la section 4.3.4, l'évaluation du travail des étudiants est un aspect important de leur formation. En effet, certains étudiants détermineront l'effort requis pour accomplir certaines tâches dans un cours selon l'impact qu'elles peuvent avoir sur leur note finale : moins une tâche a un impact sur la note finale, moins l'étudiant y investira de l'énergie. Alors, si un enseignant utilise des évaluations qui ne coïncident pas bien avec l'activité de programmation, il est probable que bien des étudiants ne programmeront pas dans le cours et ne développeront donc pas leurs compétences de programmation. Alice conclut : « Those that take the extra step will learn. Those that don't won't. »

5.2.3 La communauté mathématique

Cette idée de laisser les étudiants développer indépendamment leurs compétences en programmation correspond à une culture didactique en mathématiques. En effet, plusieurs de nos participants expliquent que c'est exactement ainsi qu'ils ont développé leurs propres compétences de programmation. Alice, par exemple, décrit même ses premières expériences avec la programmation comme résultant de sa propre initiative plutôt que de celle de ses professeurs ; la plupart des autres mathématiciens, qui ont tous suivi des cours de base en programmation pendant leur baccalauréat, situent pendant leurs études supérieures l'écriture des premiers programmes qui les ont aidés à vraiment faire des mathématiques. La tradition dans l'enseignement des mathématiques a longtemps été liée à une transmission logique et ordonnée des résultats et à l'accomplissement d'exercices qui n'impliquent pas nécessairement le soutien d'outils informatiques. Et cette culture est perpétuée par des mathématiciens qui l'ont vécue comme étudiants. Olivier dit, par exemple : « Plus ou moins j'enseigne de la façon dont on m'a enseigné il y a longtemps. » Nathan nous rappelle de plus qu'en commençant à enseigner un nouveau cours, un mathématicien le fait typiquement de la même manière que le professeur précédent, tout au moins au début. En raison de ces traditions, les étudiants au premier cycle pourraient recevoir une formation limitée en programmation ; or, une fois aux cycles supérieurs, tout professeur interviewé exigerait des étudiants la maîtrise des outils et des méthodes dont ils ont besoin. Tout comme leurs directeurs, les étudiants pourraient donc avoir à prendre en charge de leur propre développement de compétences de programmation. Deux questions importantes surgissent alors : de quelle façon cette culture didactique devrait-elle être maintenue ou transformée ? Et en répondant à cette première question, devrait-on considérer la place actuelle de la programmation dans la communauté mathématique ?

Comme la description des pratiques en recherche menée à la section 4.2 le montre, la programmation se trouve parmi les *pratiques partagées* des mathématiciens appliqués. Barbara explique : « It's absolutely indispensible for applied mathematicians. I mean, I don't think there's any applied mathematician who doesn't use the computer. » Par contre, en parlant de la place de la programmation en mathématiques pures, Philippe suggère : « Programmer c'est vraiment un outil parmi tant d'autres pour faire des mathématiques. [...] c'est une habileté qui est très utile, qui n'est pas nécessaire, mais qui est utile. » Dans la communauté de

mathématiques pures, la programmation est *admise* : elle est de plus en plus acceptée, mais il existe toujours des mathématiciens qui emploient seulement des méthodes traditionnelles. Rappelons néanmoins la nature discrète, voire même *cachée*, de la programmation dans les publications des mathématiciens purs, ce qui pourrait faire obstacle à la reconnaissance de cette activité comme étant une partie importante des mathématiques pures.

Le maintien des mathématiques pures traditionnelles

Il est intéressant de noter que tous les mathématiciens appliqués que nous avons rencontrés valorisent et expriment un profond attachement envers la culture traditionnelle en mathématiques pures. Ben explique : « I really believe in pure mathematics. So I think it's really important to learn the notion of […] definitions, theorems, proofs, and the formal side of mathematics. […] so, if I teach topology, it's 100% a pure math course, and not a hint of programming; wouldn't even be appropriate. » Cette perspective correspond généralement à deux idées : (1) la programmation est surtout utile pour explorer des applications et pour faire des calculs scientifiques liés aux problèmes réels ; et (2) les mathématiques pures traditionnelles sont « belles en elles-mêmes » et n'ont pas besoin d'être accompagnées d'une application ou d'un contexte. Remarquons qu'une telle perspective tend à ignorer l'utilisation des programmes pour développer des théories abstraites, ce que même Ben fait dans sa propre recherche. Néanmoins, cela pourrait aussi résulter d'une simple différence d'opinion sur ce qui constitue la programmation : après tout, les programmes utilisés en mathématiques appliquées sont souvent beaucoup plus vastes et plus complexes que ceux écrits par les mathématiciens qui travaillent dans des domaines plus purs.

L'ouverture aux mathématiques impliquant la programmation

Tandis que les mathématiciens appliqués de notre étude paraissent tous d'accord avec l'idée de maintenir une culture traditionnelle en mathématiques pures, quelques mathématiciens purs sont plus critiques à l'endroit de cette culture et de la façon dont elle peut brouiller les perspectives des étudiants. Nathan, par exemple, se considère comme un mathématicien pur à l'origine, mais se trouve actuellement dans un département plus appliqué. Sa position unique lui permet de remarquer que parfois « dans un programme de mathématiques, la culture est très, très puriste. Donc, les mathématiques c'est pur, c'est papier-

crayon seulement ; si on fait des applications c'est mauvais, on est moins bon, encore on est de deuxième ordre. [...] puis je trouve que ça n'aide pas les étudiants. » Nathan dit qu'en exposant tous les étudiants en mathématiques à l'utilisation de l'ordinateur pour traiter des applications, ces derniers pourraient développer une appréciation pour les types différents de mathématiques. D'autres mathématiciens purs ajoutent que, pour ouvrir les perspectives des étudiants, ils doivent aussi être exposés à l'écriture de programmes pour la résolution des problèmes purs. Philippe explique : « Il est important qu'ils voient que ce n'est pas nécessairement juste réservé aux maths appliquées [...] les étudiants, des fois ils pensent [...] que c'est toujours la rigueur, puis si je touche à l'ordinateur, c'est blasphème. [...] Et c'est tellement un handicap sérieux. » On peut se demander si une plus grande visibilité dans la communauté mathématique des programmes utilisés par les mathématiciens purs pourrait contribuer à cette ouverture des perspectives des étudiants.

Encore une fois, nous constatons que l'idéal partagé par l'ensemble de nos participants se trouve dans un certain équilibre : personne ne veut faire disparaître les mathématiques traditionnelles sans ordinateur, mais personne ne veut non plus ignorer les pratiques informatiques variées qui ont cours aujourd'hui au sein de la communauté mathématique.

5.2.4 La société

Quand on considère le contexte de la société dans son ensemble, il paraît légitime de poser la question suivante : quels types de pratique préparent le mieux les étudiants aux différentes situations qui pourraient s'offrir à eux après leur baccalauréat ? Ou, de façon plus critique, est-ce que la place de la programmation dans un premier cycle en mathématiques devrait refléter les besoins actuels du marché du travail ?

Favoriser la diversité

Plusieurs de nos participants croient que les universités devraient permettre aux étudiants de choisir parmi un ensemble varié de chemins. Par exemple, Adèle explique : « Je n'aime pas beaucoup la rigidité de mettre tout le monde dans la même moule [...] pour moi l'idéal est qu'on donne la formation de base pour que tu puisses savoir ce que tu peux choisir. Puis ensuite de ça, fais tes choix. » Barbara rappelle que plusieurs mathématiciens ont réussi

sans avoir développé d'expertise en programmation. Ben ajoute que ces mathématiciens, qui réfléchissent de façon unique, sont importants au développement des mathématiques et du monde en général. Plusieurs de nos participants soutiennent ainsi que chaque étudiant devrait avoir une expérience de base en programmation qui lui permette de prendre des décisions éclairées selon ses intérêts, ses buts et ses talents.

Quelques questions surgissent face à une telle vision. Premièrement, comment un étudiant peut-il continuer avec la programmation s'il n'y a pas beaucoup d'opportunités de le faire, par exemple, à cause du maintien de la culture didactique traditionnelle, des thèmes privilégiés au sein d'un département ou d'un manque de cours dont les objectifs puissent se prêter à l'inclusion d'activités de programmation ? Deuxièmement, les étudiants ont-ils vraiment une bonne idée de ce que sont leurs intérêts, leurs buts ou leurs talents au début de leur baccalauréat ? Ont-ils besoin de plus de temps pour développer leur maturité mathématique avant de prendre des décisions concernant leur futur ? Alain suggère qu'un premier cycle en mathématiques soit vu comme une formation de base, dans laquelle « il faut voir tous les outils de toutes les façons différentes. Après, à la maîtrise ou au doctorat, on choisit ce qu'on veut faire. Si quelqu'un veut continuer à faire de l'abstraction, il peut passer le restant de sa vie à faire de l'abstraction. Mais, il aura vu un jour, il aura fait une connexion un jour avec d'autres aspects. »

Favoriser la majorité

Un autre sous-groupe de nos participants défend plutôt l'idée que le développement de compétences de programmation pourrait être utile pour la majorité des étudiants en mathématiques. Alice, par exemple, raisonne de la façon suivante :

> If a student's going to go on to graduate work in math in a sub discipline that requires no programming, and they're convinced that after their graduate work they will continue to need no programming in their future career, then scientific computing might be a waste of time. The number of people that successfully manage to go this route is negligible. That's what the data suggests. So for the vast majority of students, they will either have to use scientific computing as part of their mathematical careers, or their other careers, whatever they may be. And so it's very useful and important for them to get comfortable with scientific computing early on. It makes it much more believable when they go for job interviews.

Dans sa description du programme idéal de premier cycle en mathématiques, Alice suggère qu'une place égale soit accordée aux activités de preuve et aux activités informatiques. Nathan et Alain mentionnent des idées similaires, mais ils mettent plus d'emphase sur un aspect qui reste subtil dans le discours d'Alice : si un étudiant veut devenir enseignant ou professeur à n'importe quel niveau, ou si un étudiant veut devenir théoricien, peut-être qu'il n'aura pas besoin de compétences de programmation. On sent, une fois de plus, l'influence des cultures traditionnelles dans la recherche mathématique et dans l'enseignement des mathématiques. Même si cette idée n'est pas complètement fausse (p.e., il existe des théoriciens qui n'utilisent aucun outil informatique), on peut se demander si ces étudiants ne pourraient pas bénéficier également d'une formation en programmation. Les théoriciens-professeurs que nous avons interviewés ont tous profité de telles compétences, soit pour résoudre des problèmes dans leur recherche, soit pour soutenir l'apprentissage de leurs étudiants. De plus, si l'on exclut systématiquement les futurs enseignants et professeurs de mathématiques des initiatives ciblées pour développer des compétences de programmation, on reporte à la génération suivante les tensions curriculaires que l'on vit aujourd'hui.

Enfin, si nos participants ne sont pas tous d'accord sur le moment adéquat pour les étudiants de commencer à faire leurs propres choix concernant leur interaction avec la programmation, ils ont en commun un engagement à vouloir aider leurs étudiants à réussir après leur baccalauréat.

6 Conclusions

Dans ce dernier chapitre, nous présentons une synthèse de nos résultats, les limites et les apports de notre étude et, enfin, quelques implications et dernières remarques.

6.1 Une synthèse des résultats

Dans ce mémoire, nous avons cherché à mieux comprendre l'écart intrigant rapporté par Buteau et coll. (2014) : si 46% des 302 mathématiciens sondés écrivent des programmes informatiques dans le cadre de leur recherche, seulement 18% d'entre eux intègrent la programmation dans leurs cours. La première donnée reflète le potentiel énorme de la programmation pour faire des mathématiques et suggère la pertinence de l'intégrer dans la formation. La seconde donnée a inspiré la problématique de notre recherche : pourquoi existe-t-il un tel écart ? Pour répondre à cette question, nous avons mis sur pied une étude qualitative exploratoire auprès de 14 mathématiciens travaillant dans des domaines variés et à différentes universités à travers le pays et ayant utilisé des programmes informatiques dans leur recherche ou dans leur enseignement. Durant nos entrevues semi-dirigées, nous avons demandé à ces chercheurs-professeurs de décrire leurs propres expériences avec la programmation ainsi que les expériences éventuelles de programmation qu'ils planifient pour leurs étudiants. Les résultats de notre analyse qualitative des transcriptions ont été présentés aux Chapitres 4 et 1.

Tout d'abord, l'ensemble des perceptions de nos participants nous a permis de circonscrire ce qu'ils considèrent comme « l'activité de programmation » et de la situer parmi les activités dites « mathématiques ». De nos rencontres, la programmation en mathématiques apparaît typiquement comme une activité comprenant trois tâches, dont l'importance relative peut varier et qui se déploient souvent de façon non linéaire : *le développement d'un algorithme, le codage de l'algorithme* et *la vérification et la validation du programme.* Selon nos participants, plusieurs caractéristiques de ces tâches rendent la programmation semblable à « faire des mathématiques » ; on a mentionné, par exemple, la similitude entre la programmation et la résolution de problèmes mathématiques ou la construction d'une preuve formelle. Néanmoins, l'activité de programmation occupe encore une place de second rang, la place d'outil, au regard des résultats mathématiques qu'elle permet de produire.

Les nombreux exemples spécifiques de programmation décrits par nos participants nous ont permis de donner un premier portrait de la place de la programmation en recherche mathématique. Dans la résolution de problèmes appliqués, la programmation est d'abord et avant tout perçue comme une nécessité pour des raisons *pragmatiques* : l'écriture des programmes sert à faire des calculs, à visualiser et à dégager de nouvelles régularités et à simuler des phénomènes qui ne seraient même pas envisageables sans l'informatique. La programmation est donc une composante des techniques *partagées* par tous les mathématiciens appliqués. Les mathématiciens purs, eux, peuvent aussi décider de créer des programmes informatiques, soit spontanés, partageables ou complexes (composante des techniques tout simplement *admises*), afin de profiter de leur valeur *épistémique* : ils programment typiquement pour explorer, expérimenter et effectuer des observations menant à des conjectures convaincantes qui méritent alors qu'on se lance dans la recherche d'une démonstration rigoureuse. Dans des cas encore plutôt rares, des programmes peuvent même être développés pour s'occuper de ces démonstrations, en partie tout au moins. Finalement, certains mathématiciens consacrent des projets au développement d'outils informatiques : ils conçoivent des algorithmes spécifiques ou des logiciels complets qui pourront être mis à contribution pour résoudre une grande classe de problèmes. Les nouveaux outils ainsi produits sont souvent partagés avec d'autres utilisateurs et requièrent, par conséquent, d'être entretenus au fil des ans selon l'évolution constante de l'informatique et les buts des utilisateurs.

Dans les articles rapportant des résultats qui dépendent largement de la programmation qui a été faite, on trouve typiquement une description des programmes utilisés. Dans plusieurs projets, toutefois, les programmes ne sont pas partagés ; parfois, c'est même toute l'activité de programmation qui demeure cachée. Au moment d'arriver aux résultats recherchés, la programmation et le programme, tout comme d'autres éléments de l'étape d'exploration, deviennent « jetables ». Quelques mathématiciens ne gardent même pas leurs propres programmes ; d'autres les gardent dans leurs bibliothèques personnelles ; et dans des cas uniques, lorsqu'un programme est particulièrement spécial ou écrit par un chercheur qui croit dans le partage d'outils avec la communauté mathématique, le chercheur peut décider de consacrer l'effort et le temps nécessaires pour en préparer la diffusion.

Les professeurs de notre étude nous ont aussi aidée à mieux cerner la place de la programmation dans l'apprentissage des étudiants de premier cycle. Dans ce contexte, nos participants invitent leurs étudiants à interagir à différents niveaux avec la programmation, soit pour mieux comprendre les théories mathématiques enseignées, soit pour développer des compétences de modélisation ou de simulation, s'approprier de méthodes de calcul ou de représentation ou même développer des compétences de programmation. Les étudiants peuvent être ainsi exposés à des pratiques qu'on retrouve en recherche : il arrive qu'ils soient guidés à développer des conjectures, à explorer des modèles mathématiques ou à analyser des méthodes numériques. Néanmoins, le manque de visibilité de la programmation que nous pourrions constater en recherche semble être transmis au contexte de l'enseignement, où les étudiants sont rarement invités à concevoir ou à écrire leurs propres programmes informatiques. Pour plusieurs des mathématiciens interviewés, leurs étudiants n'interagissent pas du tout avec l'activité de programmation, ils demeurent en position de spectateur face à l'utilisation des programmes par leur professeur ou ils utilisent des programmes déjà créés, possiblement avec des manipulations de l'interface ou des modifications du code.

Or, les professeurs rencontrés notent de nombreux avantages à impliquer les étudiants à de plus hauts niveaux d'interaction avec l'activité de programmation : selon les professeurs, un étudiant qui participe souvent à tout processus de la programmation pourrait

1) développer des compétences transférables à d'autres contextes (p.e., penser d'une façon logique, travailler avec l'abstraction, vérifier son travail) ;
2) mieux comprendre les concepts, les processus et les méthodes qu'il transpose en langages de programmation ;
3) ouvrir certaines boîtes noires dont l'utilisation est inévitable ;
4) gagner en autonomie dans le processus de l'instrumentalisation de l'ordinateur ;
5) être mis en contact avec un pan de l'activité mathématique, élargissant ainsi ses connaissances et sa vision des mathématiques ;
6) adopter les pratiques des mathématiciens professionnels, ouvrant ainsi ses possibilités de carrière ou de spécialisation après son baccalauréat.

Une question réapparaît alors : pourquoi existe-t-il encore cet écart entre la programmation en recherche et celle en apprentissage, qui ne permet pas aux étudiants de tirer pleinement parti de ces avantages ? Le discours des mathématiciens concernant les contextes institutionnels où ils travaillent et enseignent présente quelques éléments explicatifs. En particulier, nous y avons identifié un ensemble de contraintes, intimement liées, qui pourrait avoir un impact sur la décision d'un professeur d'intégrer ou de ne pas intégrer la programmation dans un cours donné.

Tout d'abord, un professeur considérera des éléments curriculaires tels *les objectifs* du cours ou du programme correspondant : selon nos participants, plus les objectifs sont abstraits, théoriques ou purs, moins l'ajout d'activités de programmation apparaît naturel ou même approprié. Ceci étant dit, même dans les cours dont les objectifs les rendent propices à l'intégration de la programmation (notamment, les cours en mathématiques appliquées), *les contraintes de temps* et *les prérequis* pèsent dans la décision d'y ajouter des activités de programmation. Quelques professeurs refusent d'exiger des étudiants de programmer si ces derniers n'ont pas déjà des compétences de base en programmation. D'autres sont découragés par le temps requis pour préparer de nouvelles activités et pour envisager leur intégration dans un curriculum déjà bien chargé et déterminé. Alors, quels sont vraiment les meilleurs endroits pour exposer les étudiants aux activités de programmation ? Ces endroits existent-ils déjà ? Ou gagnerait-on à envisager des changements curriculaires ?

En fin de compte, étant donné l'importance accordée par les universités à la *liberté académique*, un professeur peut donner un cours selon sa propre *personnalité mathématique*. Un mathématicien qui préfère des techniques à la main peut choisir de montrer aux étudiants l'utilité de ces techniques. De même, un mathématicien qui maîtrise la programmation peut partager son expertise sur différentes façons de l'utiliser en mathématiques. Quelques participants proposent comme un idéal à atteindre une formation riche où les étudiants-apprentis apprennent d'un ensemble varié d'experts, développant ainsi leur propre personnalité mathématique. Mais d'autres avertissent que si les professeurs dans un département favorisent les mêmes techniques, le résultat peut être une formation déséquilibrée, limitée, biaisée contre ou en faveur de l'intégration d'activités de programmation. Comment faire en sorte que les

étudiants soient exposés à un ensemble riche de personnalités ? À quel degré devrait-on coordonner l'intégration de la programmation sans empiéter sur la liberté académique ?

Nos participants nous rappellent également que toute formation en mathématiques est ancrée dans une *culture mathématique* selon laquelle on favorise, d'une part, la transmission de résultats et de leurs preuves formelles par une présentation magistrale et, d'autre part, la complétion à la main d'exercices qui mettent à contribution un raisonnement mathématique sans le soutien de l'ordinateur. Cette façon de faire s'est révélée efficace pour les nombreux professeurs qui, l'ayant appréciée et ayant pu se distinguer dans ce système, continuent à l'utiliser pour faire vivre à leurs étudiants les mathématiques qu'ils maîtrisent et qu'ils aiment. Elle peut conduire néanmoins à une minoration des techniques informatiques et à la délégation aux étudiants de la prise en charge de leur propre formation en programmation. Est-ce à l'étudiant de trouver les moyens de progresser en programmation ? Quels aspects de la culture traditionnelle en mathématiques pourraient faire place à une formation en programmation ?

Quelques mathématiciens suggèrent que *les ressources* disponibles (ou manquantes) dans une université peuvent aussi compliquer la situation : si l'on veut donner des cours qui intègrent des activités de programmation, il faut des professeurs qui soient prêts à préparer et à enseigner ces cours et des assistants en nombre suffisant pour évaluer les activités et les valoriser ainsi au sein du cours. Le fait d'avoir trop d'étudiants ou trop peu d'évaluateurs peut empêcher le professeur de faire bénéficier les étudiants de son cours de plus hauts niveaux d'interaction avec la programmation. Comment répondre à de telles contraintes inévitables ?

Enfin, les mathématiciens mentionnent souvent l'importance des *perspectives d'avenir des étudiants* lorsqu'ils décrivent la place idéale de la programmation dans un programme de premier cycle en mathématiques. Selon un groupe de participants, l'étudiant devrait avoir l'occasion de tracer son propre chemin selon ses intérêts, ses talents et ses buts ; selon un autre groupe, l'étudiant ne devrait pas sortir d'un baccalauréat en mathématiques sans avoir développé des compétences en programmation, compétences qui seraient appréciées sur le marché du travail actuel. Comment assurer, face aux autres contraintes institutionnelles, qu'un chemin incluant la programmation soit possible ? Quels étudiants devraient prendre un tel chemin ?

Les contraintes induites par le curriculum, le département, la communauté mathématique et la société dans son ensemble peuvent contribuer à l'écart entre la place de la programmation dans la recherche et celle dans une formation en mathématiques : si, dans la communauté de recherche, la programmation est vue comme *partagée* (communauté appliquée) ou *admise* (communauté pure), la programmation peut entrer dans les universités plutôt comme *admise* (cours/programmes appliqués) ou *contestée* (cours/programmes purs). De nouvelles questions surgissent alors : y a-t-il un intérêt à rapprocher ces visions institutionnelles de la programmation et à réduire ainsi l'écart entre la programmation des mathématiciens et celle de leurs étudiants ? Comment procéderait-on, le cas échéant ?

6.2 Les limites et apports de l'étude

Avant de nous lancer à réfléchir nous-même aux questions énoncées ci-haut, il est important de noter quelques limites et apports de notre étude.

En développant notre approche méthodologique, nous avons choisi d'interviewer un nombre restreint de professeurs qui intègrent tous la programmation dans leur travail, soit dans leur recherche, leur enseignement ou les deux contextes. Par conséquent, il serait impossible de généraliser nos résultats à tous les mathématiciens travaillant dans toutes les universités à travers le pays. Rappelons, néanmoins, que l'objectif d'une étude qualitative, comme la nôtre, n'est jamais d'arriver à de telles généralisations ; nous avons cherché plutôt à compléter des études quantitatives déjà menées (notamment celle de Buteau et coll., 2014) en donnant une description approfondie des pratiques de 14 mathématiciens et leurs étudiants. En cherchant à documenter des aspects plus spécifiques de leur pratique et des contextes dans lesquels elle s'inscrit, nous souhaitions expliquer l'écart entre l'utilisation des programmes informatiques par des mathématiciens en recherche et l'intégration de la programmation dans l'enseignement et l'apprentissage des mathématiques au niveau universitaire.

En particulier, notons que nous n'avons pas considéré la perspective du mathématicien qui n'utilise pas du tout de programmes informatiques. Ce choix méthodologique nous permettait à la fois de recueillir un ensemble d'opinions nuancées sur ce qui constitue « la programmation » en mathématiques, de donner lieu à un plus vaste ensemble d'exemples d'intégration de programmes informatiques dans la recherche et dans la formation en

mathématiques et de rapporter des perspectives sincères et critiques des contraintes institutionnelles qui paraissent limiter cette intégration. Cela dit, nous sommes toujours restée consciente du fait que nos participants ont un certain intérêt pour la programmation, lequel pourrait se traduire en un certain biais dans l'évaluation de son potentiel.

Il nous faut discuter aussi d'un autre choix méthodologique, pris pour des raisons pratiques (limite de temps, nombre de personnes impliquées dans l'étude et distribution des participants sur un vaste territoire) : nous n'avons pas effectué d'observations en classe ou lors d'activités de recherche. Nos interviewés ont décrit leurs propres pratiques en recherche ainsi que les pratiques qu'ils souhaitent aider leurs étudiants à développer par les activités de programmation qu'ils intègrent. Mais les mathématiciens font-ils exactement ce qu'ils disent ? Les étudiants développent-ils vraiment ces pratiques ? Afin de contrer au moins partiellement l'impact d'absence d'observations, nous avons employé les techniques de l'entretien d'explicitation (Vermersch, 2006) qui pourraient aider nos participants à « revivre » les exemples spécifiques dont ils ont parlé. Nous avons aussi demandé aux mathématiciens de partager des programmes informatiques et des activités d'apprentissage qu'ils ont créés et utilisés. Nous espérons ainsi avoir collecté des données riches et cohérentes, qui pourraient servir de point de départ pour des explorations ou des discussions futures.

6.3 Implications et dernières remarques

Comme nous l'avons vu, la programmation peut être une activité très mathématique. En effet, le processus de construire un programme pour accomplir une tâche donnée peut être considéré comme un cas spécifique de la résolution de problèmes mathématiques : on évalue les outils et les approches disponibles en fonction de la tâche donnée, on détermine et on organise les étapes ou les connexions logiques permettant d'arriver à la solution, on effectue soigneusement ces étapes ou ces connexions et on revient sur le produit final pour vérifier que le but a été atteint. Plus spécifiquement, le développement d'un algorithme efficace, tout comme l'écriture d'une preuve élégante, peut faire appel à un raisonnement profond, complexe, rigoureux et très mathématique, dans la construction, la validation et l'amélioration de différentes approches et structures de données. L'acte de rendre un algorithme plus abstrait, plus général, pour accomplir un ensemble plus vaste de tâches d'une même catégorie, plutôt

que seulement la tâche initialement posée, peut également être classifié comme mathématique. Même la traduction correcte d'un algorithme en un langage de programmation spécifique, qui peut être mécanique et même frustrant dans certains cas, peut demander de l'attention, de l'abstraction et de la rigueur, au même degré que la complétion et la vérification d'exercices mathématiques.

Mais être qualifié comme activité mathématique et être accepté parmi les activités importantes dans la recherche ou dans une formation en mathématiques sont deux choses différentes : la programmation devrait-t-elle toujours être confinée à n'être qu'*une approche possible pour faire des mathématiques* ? Ou est-on prêt à l'accepter comme *une façon importante de faire des mathématiques*, autant dans la poursuite de recherche mathématique que dans la formation des mathématiciens futurs ?

La reconnaissance de la programmation comme un processus mathématique important exigerait qu'on traite cette activité comme les autres activités valorisées en mathématiques, telles la preuve formelle ou la résolution des problèmes. Dans le contexte de la recherche en mathématiques, on gagnerait à assurer une meilleure visibilité de cet aspect du travail, en rendant disponibles plus de programmes, en parlant plus souvent de la programmation dans les articles auxquels elle contribue ou en discutant du processus avec des collègues. Un tel partage pourrait inspirer de nouvelles études ou collaborations dans la communauté scientifique et, par extension, le développement de nouvelles méthodes et connaissances en mathématiques. Cela pourrait aussi aider à rendre cet aspect du travail d'un mathématicien beaucoup plus accessible, étudiable et, par conséquent, enseignable. Dans le contexte de l'apprentissage des mathématiques, on pourrait simplement offrir aux étudiants plus d'occasions d'étudier des aspects de la programmation, d'écrire leurs propres programmes et d'être évalués sur cette écriture, tout comme ils étudient des méthodes de preuve ou des approches de résolution, écrivent des preuves ou résolvent des problèmes et reçoivent des évaluations correspondantes.

Évidemment, ces changements sont plus faciles à dire qu'à faire : ils requerraient beaucoup de réflexion. Par exemple, dans les universités on semble être d'accord sur l'importance d'un cours de base en programmation. Mais enseigne-t-on maintenant les aspects les plus pertinents pour une formation mathématique et ses applications ? En plus de familiariser les étudiants aux outils spécifiques, il pourrait être pertinent, comme l'ont avancé

quelques-uns de nos participants, de leur faire découvrir des concepts informatiques plus avancés (p.e., les différentes structures de données, les différents types de langage de programmation, des notions d'algorithmique) ; ceci pourrait aider les étudiants à progresser dans leur genèse instrumentale, à s'adapter plus facilement à l'évolution constante d'outils informatiques et à être en mesure de résoudre une plus grande classe de problèmes. On pourrait envisager un cours au sein du département de mathématiques, développé en collaboration avec des informaticiens, qui cherche à montrer aux étudiants le caractère et la pertinence mathématiques de ces concepts informatiques. Et si le curriculum pré-universitaire devait évoluer pour amener les élèves à développer des compétences de base en programmation[7], y compris l'utilisation d'un ou de plusieurs langages spécifiques, on pourrait pousser plus loin l'apprentissage des étudiants dans un cours « de base » au niveau baccalauréat.

Avec de bons cours de base, à l'école, au collège ou à l'université, les professeurs pourraient miser sur la capacité de leurs étudiants à programmer dans leurs cours. Mais quelques questions importantes demeurent : par exemple, quels sont les meilleurs moments, dans un premier cycle en mathématiques, pour inviter les étudiants à interagir avec des programmes informatiques ? Si la bonne réponse à cette question n'est pas claire, nos entrevues nous suggèrent quelques idées. Tout d'abord, si l'on accepte l'idée que les étudiants en mathématiques doivent développer leurs compétences de programmation et connaître la pertinence de cette activité mathématique, ils devraient vivre plus souvent et plus systématiquement des expériences de programmation. Il est fort probable qu'il existe des cours typiques (p.e., ceux de calcul, d'algèbre linéaire ou d'équations différentielles) qui bénéficieraient de l'intégration d'activités de programmation, soit dans les devoirs, dans un environnement de laboratoire ou même dans le contexte d'*une classe inversée* (voir, p.e., Love, Hodge, Grandgenett et Swift, 2014) ; dans cette dernière orchestration, les étudiants

[7] Notons, par exemple, les réflexions curriculaires qui ont cours en ce moment pour le programme de Sciences de la nature au collégial (d'après des conversations que nous avons eues avec notre directrice de recherche) et les changements récents au curriculum scolaire en Angleterre, selon lesquels les élèves devraient apprendre à programmer dès l'école primaire (McCaskill, 2013; Swidenbank, 2015).

s'approprient en dehors des heures de classe le contenu présenté à l'aide de vidéos et autres ressources mises à leur disposition et ils consacrent plutôt le temps en classe à réaliser des activités enrichissantes sous la direction du professeur. L'ajout de nouveaux cours ou programmes pourrait aussi être une voie à considérer. Nos participants ont suggéré, par exemple, le développement de cours en mathématiques expérimentales ou en modélisation interdisciplinaire qui pourraient exposer les étudiants à des activités de programmation qui s'approchent de celles de leurs professeurs.

Comme l'ajout d'expériences de programmation à travers un baccalauréat risque de bousculer certaines traditions dans l'enseignement des mathématiques, il exigerait une réflexion sérieuse. L'organisation de groupes de discussion, incluant des mathématiciens d'universités différentes, de divers domaines d'études et de plusieurs niveaux d'expertise en programmation, pourrait faciliter le partage non seulement d'activités de programmation riches en apprentissages mathématiques, mais aussi d'idées pour la mise en place des conditions nécessaires à leur intégration face aux contraintes institutionnelles, incluant des éléments d'organisation curriculaire. L'incorporation de la programmation dans un cours donné pourrait nécessiter, par exemple, un changement au contenu typiquement enseigné, aux objectifs généraux du cours et aux prérequis correspondants. De façon semblable, le développement de nouveaux cours intégrant la programmation pourrait conduire à des changements dans la structure des programmes. Notons l'importance d'inclure dans les discussions la perspective de mathématiciens aux profils variés. Ceux qui ont déjà réussi à faire des intégrations innovatrices dans leurs universités pourraient donner des conseils très utiles, basés sur des expériences concrètes. Mais ceux avec des connaissances minimales en programmation pourraient aussi apporter des perspectives importantes sur la situation. Rappelons le cas des professeurs dans notre étude qui ne programment pas dans leur recherche, mais qui, avec l'aide d'étudiants aux cycles supérieurs, ont décidé d'intégrer la programmation dans leurs cours et d'apprendre, au fur et à mesure, avec leurs étudiants.

Mais pour qu'un tel mouvement se mette en place, il convient d'abord de faire valoir l'intérêt d'aller dans cette direction, en encourageant le partage et le développement d'activités de programmation susceptibles d'être intégrées dans les cours actuels. Comme notre mémoire en témoigne, il existe déjà des exemples d'intégration fort réussis de la programmation qui

pourraient servir de point de départ à la réflexion. Considérons, par exemple, les activités décrites par nos participants où l'on donne aux étudiants des programmes déjà développés qu'ils doivent modifier selon des buts différents. Pourrait-on employer cette approche au début des cours qui dépendent d'un nouveau langage de programmation, pour aider les étudiants dans leur genèse instrumentale ? Cette technique pourrait aussi se prêter aux situations où l'on veut montrer aux étudiants un certain algorithme qu'ils auraient à décrire et à explorer seulement en regardant le code et en utilisant le programme. Des études futures pourraient évaluer la nature, l'utilité et l'efficacité des activités existantes, proposer ainsi la création d'autres activités enrichissantes et encourager enfin le développement d'un répertoire d'activités, auxquelles les chercheurs, les enseignants et les mathématiciens pourraient tous contribuer.

Si l'étude des pratiques des mathématiciens, comme nous l'avons faite dans ce mémoire, peut servir d'inspiration pour le développement de nouvelles activités d'apprentissage impliquant la programmation, on pourrait aussi gagner à mener d'autres études auprès d'un échantillon plus spécifique ou plus élargi de mathématiciens. Consacrer une étude à un champ mathématique spécifique pourrait nous fournir une compréhension plus détaillée et plus cohérente des tâches, des techniques et des justifications liées à ce champ ; on pourrait documenter, par conséquent, des exemples qui aideraient à mieux intégrer la programmation dans des cours ou des programmes spécifiques. À l'inverse, l'ajout d'expériences de mathématiciens professionnels provenant de plus de champs d'études ou d'application des mathématiques, en dehors du contexte académique, pourrait conduire à une compréhension élargie de la façon dont la programmation est impliquée dans toute activité mathématique. Cela pourrait se révéler particulièrement utile pour déterminer la place de la programmation dans un programme en mathématiques préparant aussi bien au marché du travail ou à la poursuite d'études dans un secteur d'application des mathématiques qu'à la poursuite d'études en mathématiques ou à leur enseignement. Ultimement, rassembler une communauté spécifique de chercheurs ou réunir des mathématiciens académiques avec des mathématiciens travaillant en industrie au sein d'une étude plus interactive (voir, p.e., Morrissette, 2011), pourrait aussi conduire à un partage des programmes, des techniques de programmation et des idées concernant la visibilité de la programmation dans la communauté mathématique.

Si cette étude nous a fait prendre conscience des défis entourant le projet de rendre plus visible la programmation des mathématiciens, elle nous a également convaincue de la nécessité de surmonter ces défis. Premièrement, l'activité de programmation appartient à une partie spéciale et même privée du travail des mathématiciens : la partie où ils sont en train de réfléchir, de jouer, d'explorer, de vraiment faire de la recherche. Il n'est pas toujours facile de partager cette composante personnelle du travail, avec ses parties incomplètes, peu raffinées, et de la soumettre à la critique. Mais en essayant de partager ces processus intimes, on pourrait aider ceux qui apprennent à développer une vision beaucoup plus réaliste de ce que « faire des mathématiques » implique, une vision qui prenne en compte tout l'historique du processus de construction et pas seulement les résultats retravaillés et finement polis. Deuxièmement, les processus de recherche sont généralement difficiles à saisir : ils sont souvent désordonnés et incompréhensibles pour l'observateur non informé. Pourrait-on donc transformer l'étudiant observateur, qui veut vraiment apprendre à faire des mathématiques, en participant actif ? Et troisièmement, tout comme pour les techniques de preuve ou de résolution, les techniques de programmation sont aussi diverses que les « personnalités mathématiques » mentionnées par plusieurs de nos participants. Imaginons, néanmoins, la richesse de mettre les mathématiciens et les étudiants en contact avec ces personnalités (de programmation) différentes ; cela pourrait les mener à adopter plusieurs approches et à développer plus avant leur propre personnalité mathématique.

En définitive, tous les mathématiciens que nous avons rencontrés avancent aussi la pertinence de reconnaître la diversité des mathématiques : dans un programme idéal de premier cycle, ils recherchent tous un ensemble équilibré d'expériences auxquelles soient exposés les étudiants. Les activités de programmation ne doivent pas constituer l'essentiel des cours, mais elles doivent quand même bénéficier d'une plus grande attention que celle qu'elles reçoivent aujourd'hui. Le chemin permettant d'atteindre pareil équilibre n'est pas clairement tracé et pourrait se révéler exigeant face aux nombreux défis et contraintes identifiés. Néanmoins, le rapprochement des perspectives des mathématiciens de notre étude et leur intérêt partagé à trouver de meilleures solutions suggèrent qu'une telle convergence est possible.

7 Bibliographie

Abrahamson, D., Berland, M. W., Shapiro, R. B., Unterman, J. W. et Wilensky, U. (2004). Leveraging epistemological diversity through computer-based argumentation in the domain of probability. Dans Y. B. Kafai, W. A. Sandoval, N. Enyedy, A. S. Nixon et F. Herrera (dir.), *Proceedings of The Sixth International Conference of the Learning Sciences* (p. 28 – 35). Mahwah NJ : Lawrence Erlbaum Associates.

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*, 133-169.

Artigue, M. (2002). Learning mathematics in a CAS environment: The genesis of a reflection about instrumentation and the dialectics between technical and conceptual work. *International Journal of Computers for Mathematical Learning*, *7*, 245-274.

Asiala, M., Dubinsky, E., Mathews, D. M., Morics, S. et Oktaç, A. (1997). Development of students' understanding of cosets, normality, and quotient groups. *Journal of Mathematical Behaviour, 16*(2), 241-309.

Bailey, D. H. et Borwein, J. M. (2005). Experimental mathematics: Examples, methods and implications. *Notices of the AMS, 52*(5), 502-514.

Balacheff, N. (1994). La transposition informatique. Note sur un nouveau problème pour la didactique. Dans M. Artigue et coll. (dir.), *Vingt ans de Didactique des Mathématiques en France* (p. 364-370). Grenoble : La Pensée Sauvage.

Blum, W. et coll. (2002). ICMI Study 14: Applications and modelling in mathematics education - discussion document. *Educational Studies in Mathematics, 51,* 149-171.

Buteau, C., Jarvis, D. et Lavicza, Z. (2014). On the integration of computer algebra systems (CAS) by Canadian mathematicians: Results of a national survey. *Canadian Journal of Science, Mathematics, and Technology Education*, *14*(1), 1-23.

Buteau, C. et Muller, E. (2010). Student development process of designing and implementing exploratory and learning objects. Dans *Proceedings of the Sixth Conference of European Research in Mathematics Education* (p. 1111-1120). Lyon, France.

Caron, F. (2001). *Effets de la formation fondamentale sur les compétences d'étudiants universitaires dans la résolution de problèmes de mathématiques appliquées* (thèse inédite). Université de Montréal, Montréal.

Caron, F. et Squalli, H. (2009). *Report of Working Group 6: Mathematical modelling and science.* Répéré à https://cms.math.ca/Events/CMEF2009/reports/wg6-report.pdf

Chevallard, Y. (1998). *Analyse des pratiques enseignantes et didactique des mathématiques : L'approche anthropologique*. Repéré à http://yves.chevallard.free.fr/spip/spip/IMG/pdf/Analyse_des_pratiques_enseignantes.pdf

Chevallard, Y. (2002). *Organiser l'étude : 3. Écologie & régulation*. Repéré à http://yves.chevallard.free.fr/spip/spip/IMG/pdf/Organiser_l_etude_3.pdf

Churchhouse, R. F. et coll. (dir.). (1986). The influence of computers and informatics on mathematics and its teaching. Dans *ICMI Study Series* (vol. 1). Cambridge, England : Cambridge University Press.

Cornu, B. (1992). Évolution des mathématiques et de leur enseignement. Dans B. Cornu (dir.), *L'ordinateur pour enseigner les mathématiques* (p. 147-159). Paris : Presses Universitaires de France.

Feurzeig, W. et Papert, S. A., avec une préface par Lawyler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487-501.

Fortin, A. (2011). *Analyse numérique pour ingénieurs.* Presses Internationales Polytechnique.

Guntamukkala, V., Wen, H. J. et Tarn, J. M. (2006). An empirical study of selecting software development life cycle models. *Human Systems Management, 25*, 265-278.

Lavicza, Z. (2010). Integrating technology into mathematics teaching at the university level. *ZDM The International Journal on Mathematics Education*, *42*(1), 105-119.

Leron, U. et Dubinsky, E. (1995). An abstract algebra story. *The American Mathematical Monthly*, *102*(3), 227-242.

Love, B., Hodge, A., Grandgenett, N. et Swift, A. W. (2014). Student learning and perceptions in a flipped linear algebra course. *International Journal of Mathematical Education in Science and Technology, 45*(3), 317-324.

Marshall, N. (2012). *Contextualizing the learning activity of designing and experimenting with interactive, dynamic mathematics exploratory objects* (mémoire inédit). Brock University, St. Catharines.

Marshall, N., Buteau, C., Jarvis, D. et Lavicza, Z. (2012). Do mathematicians integrate computer algebra systems in university teaching? Comparing a literature review to an international survey study. *Computers & Education*, *58*, 423-434.

McCaskill, S. (2013, 8 juillet). New national curriculum to teach five year olds computer programming: Education Secretary Michael Gove to implement new computing curriculum by September 2014. *TechWeekEurope*. Répéré à http://www.techweekeurope.co.uk/news/national-curriculum-ict-education-computing-121214

Morrissette, J. (2011). Vers un cadre d'analyse interactionniste des pratiques professionnelles. *Recherches Qualitatives, 30*(1), 10-32.

Muller, E., Buteau, C., Ralph, B. et Mgombelo, J. (2008). Learning mathematics through the design and implementation of exploratory and learning objects. *International Journal for Technology in Mathematics Education , 16* (2), 63-74.

Paillé, P. et Mucchielli, A. (2010). *L'analyse qualitative en sciences humaines et sociales*. Paris : Armand Colin.

Pesonen, M. et Malvela, T. (2000). A reform in undergraduate mathematics curriculum: More emphasis on social and pedagogical skills. *International Journal of Mathematical Education in Science and Technology, 31*(1), 113-124.

Polya, G. (1945). *How to Solve It.* Princeton, NJ : Princeton University Press.

Robert, Paul. (2010). *Le Nouveau Petit Robert de la langue française.* Paris : Le Robert.

Schoenfeld, A. H. (1985). *Mathematical Problem Solving.* Orlando, FL : Academic Press, Inc.

Society for Industrial and Applied Mathematics. (2012). *Mathematics in industry*. Répéré à http://www.siam.org/reports/mii/2012/report.pdf

Swidenbank, R. (2015, 15 janvier). Coding in British schools: A review of the first term. *Computerworld UK.* Répéré à http://www.computerworlduk.com/careers/coding-in-british-schools-review-of-first-term-3595505/

Trouche, L. (2000). La parabole du gaucher et de la casserole à bec verseur : Étude des processus d'apprentissage dans un environnement de calculatrices symboliques. *Educational Studies in Mathematics*, *41*, 239-264.

Trouche, L. (2004). Managing the complexity of human/machine interactions in computerized learning environments: Guiding students' command process through instrumental orchestrations. *International Journal of Computers for Mathematical Learning, 9,* 281-307.

Van der Maren, J.-M. (1996). Le codage et le traitement des données. Dans *Méthodes de recherche pour l'éducation* (p. 427-457). Montréal/Bruxelles : PUM et de Boeck.

Vérillon, P. et Rabardel, P. (1995). Cognition and artifacts: A contribution to the study of thought in relation to instrumented activity. *European Journal of Psychology of Education, 10*(1), 77-101.

Vermersch, P. (2006). *L'entretien d'explicitation*. Paris, France : ESF éditeur.

Wilensky, U. (1995). Paradox, programming, and learning probability: A case study in a connected mathematics framework. *Journal of Mathematical Behavior*, *14*(2), 253-280.

Annexe A : Courriel d'invitation

RE : L'utilisation de la programmation informatique par les mathématiciens - Invitation à participer à une étude

Chère /Cher Madame/Monsieur (nom de famille),

Je m'appelle Laura Broley et je suis étudiante à la Maîtrise en mathématiques à l'Université de Montréal sous la direction d'Yvan Saint-Aubin (Département de mathématiques et de statistique) et de France Caron (Département de didactique).

Je vous envoie ce courriel pour solliciter votre participation à une étude concernant l'utilisation de la programmation informatique par les mathématiciens dans leur vie professionnelle. Votre participation à l'étude serait limitée à une seule entrevue, d'une durée d'une à deux heures, qui aurait lieu pendant l'été ou l'automne 2014.

Ce serait merveilleux si vous acceptiez de partager vos expériences avec moi ! Je vous remercie de votre attention.

Sincères salutations,

Laura Broley

Annexe B : Guide d'entrevue

INTRODUCTION :

- But de l'entrevue
- Formulaire de consentement
- Questions ?
- Test audio
- Acceptez-vous de commencer ?

- Voulez-vous commencer par me parler un peu de vos intérêts de recherche ?
- Quels cours enseignez-vous ?
- Où utilisez-vous la programmation dans votre vie professionnelle ?

PROGAMMATION :

HISTOIRE :

- Première expérience avec la programmation
- D'autres expériences importantes ?

RECHERCHE :

- Avez-vous déjà envisagé d'utiliser la programmation dans votre recherche ?

OU

- Un moment marquant
- Un morceau de code
- Utilisation typique de la programmation
- Évolution de l'utilisation typique
- Quand avez-vous commencé à utiliser la programmation dans votre recherche ?
- Faites-vous mention de votre programmation dans vos publications ?
- Comment la programmation est-elle vue dans votre domaine de recherche ? Dans d'autres domaines ?
- Rôle futur de la programmation

ENSEIGNEMENT

- Avez-vous déjà envisagé d'intégrer ou d'utiliser la programmation dans vos cours ?

OU

- Un moment marquant
- Une activité ou une présentation
- Intégration typique de la programmation : Utilisation par le professeur ? Par les étudiants ? Évaluation ?
- Évolution de l'intégration typique
- Quand avez-vous commencé à intégrer la programmation dans votre enseignement ?
- Quelle est la place de la programmation dans ce département ? Est-ce qu'on parle de l'intégration de la programmation ?
- Dans quels cours la programmation est-elle particulièrement utile ? Pas utile ?
- Rôle futur de la programmation

RECHERCHE ET ENSEIGNEMENT

- Les étudiants que vous supervisez doivent-ils avoir des compétences en programmation ? Comment utilisent-ils la programmation ?
- Quelle est l'intersection entre votre utilisation de la programmation dans votre recherche et votre intégration de la programmation dans votre enseignement ?
- Comment comparez-vous ce que vous faites en recherche et ce que vos étudiants font lorsqu'ils apprennent des mathématiques dans vos cours ?

QUESTION GÉNÉRALES :

- Comment décririez-vous la relation entre programmer et faire des mathématiques ?
- Comment réconciliez-vous la quête de rigueur mathématique et l'utilisation de la programmation ? À quelles conditions les étudiants ou les chercheurs peuvent-ils utiliser des boîtes noires ?
- Rêvons un peu : Quel serait votre programme idéal de premier cycle en mathématiques et quelle serait la place de la programmation dans cet idéal ?

CONCLUSION :

- Avez-vous d'autres choses à ajouter ?
- Avez-vous des commentaires au regard du déroulement de l'entrevue ?

MERCI!

Annexe C : Formulaire de consentement

Titre de la recherche : *La programmation dans le travail du mathématicien : les liens entre son utilisation en recherche et en enseignement*

Chercheuse : Laura Broley, étudiante à la maîtrise, Département de mathématiques et de statistique, Université de Montréal

Directeurs de recherche : France Caron, professeure agrégée, Département de didactique, Faculté des sciences de l'éducation, Université de Montréal ; Yvan Saint-Aubin, professeur titulaire, Département de mathématiques et de statistique, Faculté des arts et des sciences, Université de Montréal

A) RENSEIGNEMENTS AUX PARTICIPANTS

Objectifs de la recherche : Ce projet vise à mieux comprendre la place, le rôle, le potentiel et les contraintes de la programmation dans la recherche et dans l'enseignement des mathématiques au niveau universitaire, ainsi que les liens entre l'utilisation de la programmation en recherche et en enseignement.

Participation à la recherche : Votre participation à ce projet consiste à accorder une entrevue d'une à deux heures à la chercheuse qui vous demandera des éléments d'information sur l'évolution de vos pratiques liées à votre utilisation de la programmation dans votre recherche et/ou dans votre enseignement. Cette entrevue sera enregistrée sur support audio, avec votre autorisation, afin d'en faciliter ensuite la transcription. On vous demandera de partager en entrevue des extraits du code que vous avez écrit, des activités didactiques que vous avez utilisées et d'autres documents liés à votre utilisation de la programmation. Il est aussi possible qu'on vous demande l'accès à une version imprimée ou électronique de certains extraits de ces documents pour en faciliter l'analyse. Le lieu et le moment de l'entrevue seront déterminés avec la chercheuse, selon vos disponibilités et les siennes.

Confidentialité : Les observations et enregistrements effectués dans le cadre de ce projet de recherche demeureront strictement confidentiels et l'anonymat des participants sera protégé dans toute publication ou conférence rapportant les résultats de ce projet. Chaque participant à la recherche se verra attribuer un numéro et seule la chercheuse et son équipe de recherche auront la liste des participants et des numéros correspondants. De plus, aucun extrait du code ou des documents que vous aurez fournis n'apparaîtra dans les publications rapportant les résultats de ce projet sans votre autorisation explicite et préalable. Les données seront conservées toujours dans un lieu sûr. Les enregistrements seront transcrits et seront détruits, ainsi que toute information personnelle, sept ans après la fin du projet. Seules les données ne permettant pas de vous identifier seront conservées après cette période.

Avantages et inconvénients : En participant à cette recherche, vous pourrez contribuer à une meilleure compréhension des raisons qui motivent les mathématiciens à utiliser ou à ne pas utiliser la programmation dans leur recherche et/ou dans leur enseignement. Il n'y a pas de risque particulier à participer à ce projet.

Retrait du projet : Votre participation à ce projet est entièrement volontaire et vous pouvez à tout moment vous retirer de la recherche sur simple avis verbal et sans devoir justifier votre décision. Si vous décidez de vous retirer de la recherche après l'entrevue, vous pouvez communiquer avec la chercheuse par courriel. À votre demande, tous les renseignements qui vous concernent pourront être détruits. Cependant, après le déclenchement du processus de publication, il sera impossible de détruire les analyses et les résultats portant sur vos données.

B) CONSENTEMENT

Je déclare avoir lu et compris les termes du présent formulaire. Je consens librement à prendre part à cette recherche et je sais que je peux me retirer en tout temps sans avoir à justifier ma décision.

Je consens à ce que l'entrevue soit enregistrée : Oui □ Non □

Signature : _____ Date : _____

Nom : _____ Prénom : _____

La chercheuse affirme avoir fourni toutes les informations concernant ce projet et est disponible pour répondre à toute question.

Signature de la chercheuse : _____ Date : _____

Nom : _____ Prénom : _____

Pour toute question relative à l'étude, ou pour vous retirer de la recherche, veuillez communiquer avec Laura Broley.

Toute plainte relative à votre participation à cette recherche peut être adressée à l'ombudsman de l'Université de Montréal au numéro de téléphone (514) 343-2100 ou à l'adresse courriel ombudsman@umontreal.ca (l'ombudsman accepte les appels à frais virés).

Annexe D : Liste des codes et exemples d'analyse

description des pratiques
impact des contextes institutionnels
réflexions plus générales

CODE	DÉFINITION
A : pratiques	Les moments importants que les mathématiciens ont vécus lorsqu'ils étaient étudiants.
R : pratiques	Les tâches, les techniques et les justifications qui forment les pratiques en recherche.
R : pratiques souhaitées	Les tâches, les techniques et les justifications proposées pour décrire les pratiques idéales en recherche.
S : pratiques	Les tâches, les techniques et les justifications qui forment les pratiques des étudiants supervisés.
R : discipline	L'impact perçu du type de recherche (p.e., mathématiques pures, mathématiques appliquées) sur les pratiques.
R : département/université	Les normes d'un département ou d'une université et leur impact perçu sur les pratiques en recherche.
R : communauté mathématique	Les normes de la communauté mathématique et leur impact perçu sur les pratiques en recherche.
E : pratiques	Les tâches, les techniques et les justifications qui forment les pratiques en apprentissage.
E : pratiques souhaitées	Les tâches, les techniques et les justifications proposées pour décrire les pratiques idéales en apprentissage.
E : curriculum	L'impact perçu des éléments curriculaires (p.e., la limite du temps, l'environnement physique, les ressources disponibles, les types de cours, les objectifs d'un cours ou d'un programme, les prérequis d'un cours) sur les pratiques en apprentissage.
E : département/université	Les normes d'un département ou d'une université et leur impact perçu sur les pratiques en apprentissage.
E : communauté mathématique	Les normes de la communauté mathématique et leur impact perçu sur les pratiques en apprentissage.
E : société	L'impact perçu de la société (p.e., les emplois disponibles sur le marché du travail) sur les pratiques en apprentissage.
DÉF : apprendre/enseigner/chercher	Les définitions de et les relations entre « apprendre », « enseigner » et « chercher ».
DÉF : programmer/faire des maths	Les définitions de et les relations entre « programmer » et « faire des mathématiques ».

Exemples d'extraits codés et de caractérisations supplémentaires des pratiques

La programmation je l'utilise peu. Puis je me sens toujours un peu gêné par rapport à ça, surtout ici. J'ai des collègues qui sont beaucoup plus appliqués que moi, les mathématiciens très appliqués, eux utilisent la programmation et l'ordinateur beaucoup. En fait, je suis même souvent l'objet de rigolade un peu à quel point je ne suis pas très bon à ces choses-là, comparé à eux. [...] je l'ai utilisée une fois dans la recherche, une fois où j'ai pris l'initiative pour l'utiliser, puis ça m'a aidé à débloquer un résultat théorique. [...] j'avais une certaine famille de fonctions et je voulais beaucoup qu'elles aient une certaine propriété [...] je voulais qu'elles soient concaves et convexes dans certaines régions. Donc, j'ai finalement pris des exemples, je les ai programmés, des exemples très simples, et j'ai essayé de voir si la fonction avait ces propriétés-là de façon asymptotique [...] j'ai dessiné le graphe, j'ai essayé de voir, joué un peu avec les paramètres, ça semblait toujours vrai. Donc, avec des études très naïves-là, des fonctions simples, je me suis dit, « Bien, je pense que je peux croire que c'est vrai ; donc je me mets attaquer théoriquement à le prouver. »	**R : discipline** **R : pratiques** **Tâche :** vérification d'une propriété d'une famille de fonctions **Technique :** écrire un programme qui dessine le graphe des fonctions appartenant à cette famille ; changer les paramètres pour observer le comportement sur plusieurs exemples **Justification :** (*épistémique*) l'observation de plusieurs exemples qui corroborent l'hypothèse donne la confiance nécessaire pour s'engager dans la recherche d'une preuve

the New York Stock Exchange is roughly 3000 stocks, go back 252 days, that's 3000 times 252 numbers. How do you condense that into something that a human being could use? [...] I thought of something unique [...] so you think of the stocks, the 3000 stocks, they're all moving together or separately, right? But there's so much connection between them, they're very highly correlated. So how many truly different independent stocks are there? [...] this analysis takes roughly 12 hours, I do it every weekend, of computer time. [...] You could not do this without computers. There will never be an analytic formula for this. It is entirely by simulation. [...] And I'm very interested in mathematical phenomena that can only be explored using simulations and computers. The culture of mathematics is very much built around concrete formulas, analytic solutions to problems. And in my	**R : pratiques** **Tâche :** calcul du nombre de titres indépendants sur le marché boursier **Technique :** utiliser un programme informatique ; algorithme exact inconnu **Justification :** (*pragmatique*) c'est tout simplement impossible sans l'ordinateur ; il y a trop de calculs pour les faire autrement ; de plus, il n'y a pas de solutions analytiques au problème **R : communauté mathématique**

opinion, that's been limiting the development of mathematics, and holding us back. And that culture's perpetuated in our classrooms. And I think it's really unfortunate. There's certainly a place for analytic solutions, and they are beautiful and so on. But I think we have to now open our eyes to mathematics being much more than just solving algebraic equations [...] the world is simply too complex for that.	**E : communauté mathématique**

I will often present [...] pictures that I've drawn using *Mathematica* and use the 3d option to drag the objects [...] Seeing things in motion is extremely useful to be able read things properly, rather than having five pictures of the same object at different points of view. That doesn't do the same thing as actually dragging the object in front of the students. But I will not write in any of my assignments, "Write a program to do X." Never, ever. It's on your own. And I declare when I hand out the first assignment that [the students are] entitled to use any sort of programming that they want. But I want the source with comments. [...] the code has to be provided so that we can know it's original. [...] Some of my assignments are simply too dreadful to do without *Mathematica*, *Maple*, or something like that. [...] before there was no systematic teaching of *Mathematica* you could not rely on saying, "You don't want to use it? That's your problem!" But now, I can say that without any hesitation. "You despise Mathematica? Here's your pen!" It's not because they haven't been taught it. It's because they think it's beyond them, or it's beneath them. That's fine. Just prove that you can live without it.	**E : pratiques** **Tâche :** réception de théorie **Technique :** observer la manipulation d'objets au niveau de l'interface d'un programme *Mathematica* **Justification :** (*épistémique*) on peut mieux comprendre des objets animés **E : pratiques** **Tâche :** (1) réalisation des calculs ; (2) préparation des devoirs **Technique :** (1) choisir la méthode appropriée, dont la programmation est une possibilité ; (2) remettre le code **Justification :** (1) (*pragmatique*) questions trop difficiles sans l'ordinateur ; (2) besoin de prouver l'authenticité du travail **E : curriculum**

Annexe E : Code pour calculer la « variété » d'une orbite

(*le code suivant peut être exécuté en *Mathematica)**

(*déterminer la précision*)

```
$MinPrecision = 16;
```

(*définir les variables*)

```
m = 0.012149999983; (*mass ratio in Earth-Moon system*)
maxnorm = 0.001; (*max distance of perturbation from point on orbit*)
deltat = 0.01; (*time step between points on a trajectory*)
numinitialpoints = 1000; (*number of trajectories*)
maxnumtimesteps = 1000; (*max number of points on trajectories*)

(*stopping criterion for trajectories: when the x-coordinate becomes xcoorrequired the number of times represented by numtimesxcoor*)
xcoorrequired = 0;
numtimesxcoor = 2;
```

(*les données pour une solution périodique (une orbite) pour déterminer les points initiaux*)

```
q = {-0.082588062519, 0.21382188915, 0.13946316134, 0.85990162347, -0.080907690403, 0.14680913534}; (*point on orbit*)
r = {0.035208097959, -0.0034244976951, -0.022396293700, 0.015808370554, 0.0068614515560, 0.011270120883};
(*eigenvector: direction from q where initial points will lie*)
floquet = 34.21793803; (*floquet multiplier*)
s = -r/Norm[r]; (*normalized eigenvector*)
stepsize = (maxnorm - maxnorm/floquet)/numinitialpoints; (*space between initial points*)
```

(*le système d'équations*)

```
f = Function[{u, v, w, x, y, z}, {2*v + x - (1 - m)*(x + m)/Sqrt[(x + m)^2 + y^2 + z^2]^3 - m*(x - 1 + m)/Sqrt[(x - 1 + m)^2 + y^2 + z^2]^3, -2*u + y - (1 - m)*y/Sqrt[(x + m)^2 + y^2 + z^2]^3 - m*y/Sqrt[(x - 1 + m)^2 + y^2 + z^2]^3, -(1 - m)*z/Sqrt[(x + m)^2 + y^2 + z^2]^3 - m*z/Sqrt[(x - 1 + m)^2 + y^2 + z^2]^3, u, v, w}];
```

```
(*le calcul, en utilisant Runge-Kutta d'ordre quatre*)

p = ConstantArray[0, {numinitialpoints + 1, maxnumtimesteps + 1, 6}];
(*array for keeping track of trajectories*)

xcoorachieved = ConstantArray[0, {numinitialpoints + 1}];
(*array for determining when stopping criterion achieved*)

jacobi = ConstantArray[0, {numinitialpoints + 1, maxnumtimesteps + 1}];
(*array for keeping track of jacobi constant, the conserved energy E*)

For[i = 1, i < (numinitialpoints + 2), i++,

p[[i, 1, 1]] = q[[1]] + (maxnorm/floquet + stepsize*(i - 1))*s[[1]]; (*initial condition*)

p[[i, 1, 2]] = q[[2]] + (maxnorm/floquet + stepsize*(i - 1))*s[[2]];

p[[i, 1, 3]] = q[[3]] + (maxnorm/floquet + stepsize*(i - 1))*s[[3]];

p[[i, 1, 4]] = q[[4]] + (maxnorm/floquet + stepsize*(i - 1))*s[[4]];

p[[i, 1, 5]] = q[[5]] + (maxnorm/floquet + stepsize*(i - 1))*s[[5]];

p[[i, 1, 6]] = q[[6]] + (maxnorm/floquet + stepsize*(i - 1))*s[[6]];

counter = 0;

xcoorachievedcount = 0;

While[xcoorachievedcount < numtimesxcoor && counter < maxnumtimesteps - 1,

(*compute each trajectory and the related jacobi constant values*)

 counter = counter + 1;

 prevu = p[[i, counter, 1]];

 prevv = p[[i, counter, 2]];

 prevw = p[[i, counter, 3]];

 prevx = p[[i, counter, 4]];

 prevy = p[[i, counter, 5]];

 prevz = p[[i, counter, 6]];

 jacobi[[i, counter]] = 2*(-(prevu^2 + prevv^2 + prevw^2)/2 + (1/2*(prevx^2 + prevy^2) + (1
- m)/Sqrt[(prevx + m)^2 + prevy^2 + prevz^2] +m/Sqrt[(prevx - 1 + m)^2 + prevy^2 +
prevz^2]) + m*(1 - m)/2);

 k1 = f[prevu, prevv, prevw, prevx, prevy, prevz];

 k2 = f[prevu + deltat/2*k1[[1]], prevv + deltat/2*k1[[2]], prevw + deltat/2*k1[[3]], prevx +
deltat/2*k1[[4]], prevy + deltat/2*k1[[5]], prevz + deltat/2*k1[[6]]];

 k3 = f[prevu + deltat/2*k2[[1]], prevv + deltat/2*k2[[2]], prevw + deltat/2*k2[[3]], prevx +
deltat/2*k2[[4]], prevy + deltat/2*k2[[5]], prevz + deltat/2*k2[[6]]];

 k4 = f[prevu + deltat*k3[[1]], prevv + deltat*k3[[2]], prevw + deltat*k3[[3]], prevx +
deltat*k3[[4]], prevy + deltat*k3[[5]], prevz + deltat*k3[[6]]];

 p[[i, counter + 1, 1]] = p[[i, counter, 1]] + deltat/6*(k1[[1]] + 2*k2[[1]] + 2*k3[[1]] + k4[[1]]);
```

```
p[[i, counter + 1, 2]] = p[[i, counter, 2]] + deltat/6*(k1[[2]] + 2*k2[[2]] + 2*k3[[2]] + k4[[2]]);
p[[i, counter + 1, 3]] = p[[i, counter, 3]] + deltat/6*(k1[[3]] + 2*k2[[3]] + 2*k3[[3]] + k4[[3]]);
p[[i, counter + 1, 4]] = p[[i, counter, 4]] + deltat/6*(k1[[4]] + 2*k2[[4]] + 2*k3[[4]] + k4[[4]]);
(*keep track number of times required x-coordinate achieved*)
If[Mod[xcoorachievedcount, 2] == 0, If[p[[i, counter + 1, 4]] < xcoorrequired && p[[i, counter,
4]] >= xcoorrequired , xcoorachievedcount = xcoorachievedcount + 1 ], If[p[[i, counter + 1, 4]]
> xcoorrequired && p[[i, counter, 4]] <= xcoorrequired, xcoorachievedcount =
xcoorachievedcount + 1]];
  p[[i, counter + 1, 5]] = p[[i, counter, 5]] + deltat/6*(k1[[5]] + 2*k2[[5]] + 2*k3[[5]] + k4[[5]]);
  p[[i, counter + 1, 6]] = p[[i, counter, 6]] + deltat/6*(k1[[6]] + 2*k2[[6]] + 2*k3[[6]] + k4[[6]]);
  ];
xcoorachieved[[i]] = counter + 1;
counter = counter + 1;
prevu = p[[i, counter, 1]];
prevv = p[[i, counter, 2]];
prevw = p[[i, counter, 3]];
prevx = p[[i, counter, 4]];
prevy = p[[i, counter, 5]];
prevz = p[[i, counter, 6]];
jacobi[[i, counter]] = 2*(-(prevu^2 + prevv^2 + prevw^2)/2 + (1/2*(prevx^2 + prevy^2) + (1
- m)/Sqrt[(prevx + m)^2 + prevy^2 + prevz^2] + m/Sqrt[(prevx - 1 + m)^2 + prevy^2 +
prevz^2]) + m*(1 - m)/2);
 ];
```

(*les graphiques : la variété de l'orbite et la constante E de l'énergie*)

```
For[i = 1, i < (numinitialpoints + 2), i++,
l1[i] = p[[i, 1 ;; xcoorachieved[[i]], 4 ;; 6]];
g1[i] = Graphics3D[{Thickness[0.003], Hue[i/numinitialpoints], Line[l1[i]]}];
g2[i] = ListLinePlot[jacobi[[i, 1 ;; maxnumtimesteps]], PlotStyle -> {Hue[i/numinitialpoints]},
PlotRange -> {{0, maxnumtimesteps}, {-100, 40}}];
 ];
createlisttoplot1 = Table[g1[i], {i, numinitialpoints + 1}];
createlisttoplot2 = Table[g2[i], {i, numinitialpoints + 1}];
Show[Graphics3D[{Gray, Sphere[{-m, 0, 0}, 0.05], Sphere[{1 - m, 0, 0}, 0.0135]}],
createlisttoplot1, Axes -> True, AxesLabel -> {x, y, z}, Boxed -> False]
Show[createlisttoplot2, Axes -> True]
```

Annexe F : Une activité à l'aide de la programmation

Cette annexe présente un exemple d'activité d'apprentissage à l'aide de la programmation. Inspirée de l'illustration faite par Pierre dans un cours de calcul (voir la page 82), l'activité porte sur le théorème de Taylor formulé ci-dessous :

> Soient $k \geq 1$ un entier et $f : R \rightarrow R$ une fonction qui est différentiable k fois au point $a \in R$. Alors, il existe une fonction $h_k : R \rightarrow R$ telle que
>
> $$f(x) = \sum_{j=0}^{k-1} \frac{f^{(j)}(a)}{j!}(x-a)^j + h_k(x)(x-a)^k$$
>
> et
>
> $$lim_{x \to a}\, h_k(x) = 0.$$
>
> La fonction
>
> $$T_n(x) = \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j$$
>
> est appelée le n-ième polynôme de Taylor pour $f(x)$ autour de a.

Le contexte et l'orchestration

L'activité suivante pourrait se donner dans n'importe quel cours où les étudiants apprennent le théorème de Taylor pour la première fois (p.e., au collégial ou pendant la première année d'un baccalauréat). Avant de commencer l'activité, les étudiants devraient avoir une connaissance de base du théorème et d'un logiciel de calcul symbolique comme *Mathematica*. Nous proposons que l'activité soit effectuée dans un environnement de laboratoire où le professeur servirait simplement de guide et les étudiants pourraient travailler en groupe, partager leurs approches et discuter de leurs observations. Nous estimons qu'un étudiant pourrait compléter l'activité en une heure (plus ou moins). Mais évidemment, la durée requise pourrait changer selon les connaissances du groupe d'étudiants spécifique.

L'activité

1) Prenez $f(x) = \text{Sin}[x]$ et calculez le polynôme de Taylor pour $f(x)$ autour de $a = 0$ lorsque $n = 1$, $n = 5$ et $n = 10$ en utilisant l'expression du polynôme donnée dans l'énoncé du théorème (les fonctions *Mathematica* « Sum » et « Derivative » pourraient être utiles). Vérifiez que vous avez programmé le polynôme de Taylor correctement en le calculant à la main, pour les mêmes valeurs de n.

2) Gardez $f(x) = \text{Sin}[x]$ et $a = 0$. Représentez $f(x)$ et $T_n(x)$ sur le même graphique pour $n = 1$, $n = 5$ et $n = 10$ (utilisez la fonction « Plot »). Pour chaque n, où l'approximation de $f(x)$ par $T_n(x)$ est-elle la meilleure ? Quelle partie du théorème l'explique ?

3) Gardez $f(x) = \text{Sin}[x]$ et $a = 0$. Augmentez n, regardez votre graphique et commentez la relation entre $T_n(x)$ et $f(x)$. Qu'est-ce qui se passe quand $n \to \infty$?

4) Prenez $f(x) = \text{ArcTan}[x]$, $a = 0$ et $n = 1$. Existe-t-il un n tel que $|T_n(x) - f(x)| < 0{,}1$ pour tout x dans $[-0{,}5, 0{,}5]$? $[-1{,}5, 1{,}5]$? $[-5, 5]$? Comparez le comportement de $T_n(x)$ dans ce cas quand $n \to \infty$ avec ce que vous avez observé à la question précédente. Justifiez vos observations.

5) Gardez $f(x) = \text{ArcTan}[x]$ et $a = 0$. Prenez n grand (p.e., $n = 30$). Est-ce que le domaine de convergence est symétrique par rapport à a ? Est-ce le cas pour tout a ? Vérifiez-le en changeant la valeur de a (essayez, p.e., $a = \pm 1, \pm 10, \pm 100$). Décrivez ce que vous observez.

6) Prenez $f(x) = 3 - 3x + x^3 + 5x^4$ et $a = 0$. Pouvez-vous prédire le comportement de $T_n(x)$ dans ce cas lorsque vous augmenterez n ? Calculez et représentez graphiquement $T_n(x)$ pour $n = 1, 2, \ldots, 10$. Pour quelles valeurs de n l'approximation de $f(x)$ par $T_n(x)$ est-elle la meilleure ? Y a-t-il une relation entre ce que vous observez et le degré du polynôme f ?

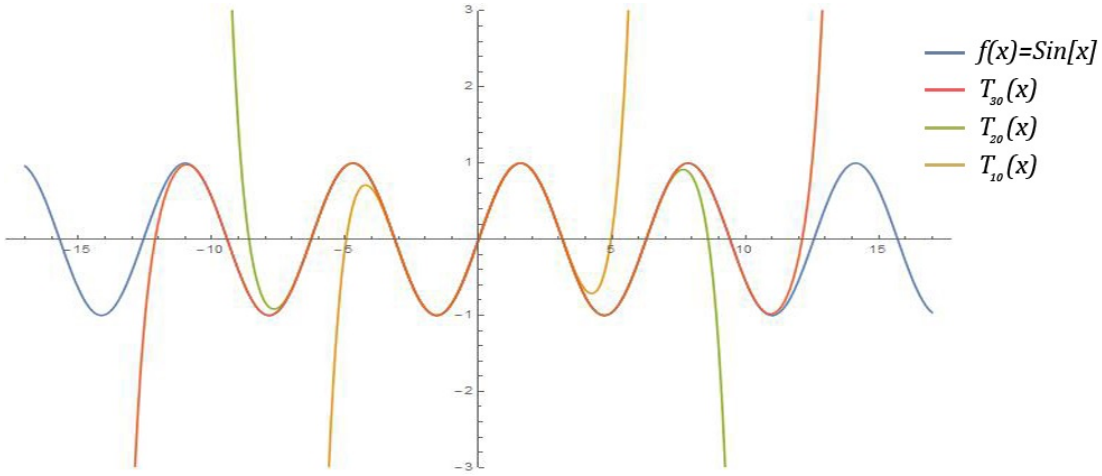
7) Prenez $a = 1$ dans la question précédente. Même question. Discutez de ce que vous trouvez.

Les buts

Cette activité vise principalement une compréhension approfondie du théorème de Taylor : son énoncé, son utilité et ses limites. Immédiatement, on demande aux étudiants d'interpréter les différents aspects du théorème en traduisant l'expression mathématique du polynôme de Taylor en langage informatique. Ensuite, à la question 2, on souhaite faire découvrir aux étudiants une partie fondamentale du théorème :

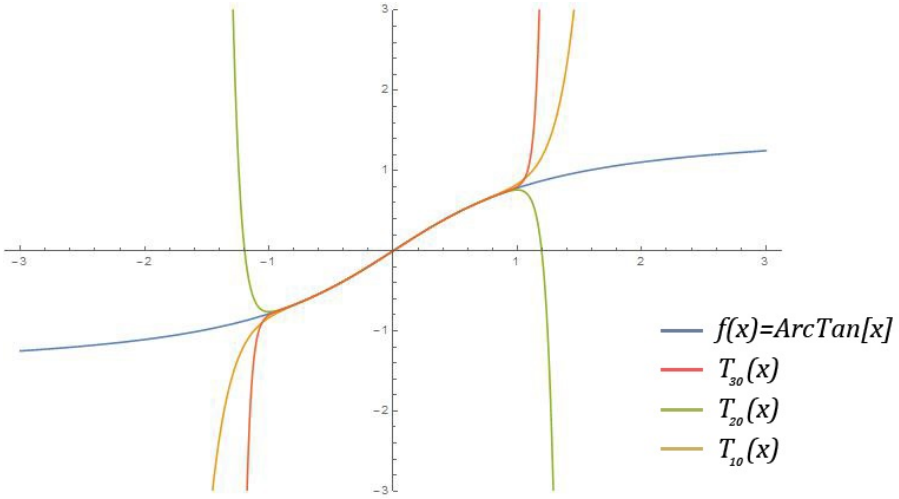
$$lim_{x \to a}\, h_k(x) = 0.$$

Autrement dit, $T_n(x)$ sert d'une bonne approximation de $f(x)$ autour de a. La troisième question expose les étudiants à un exemple « idéal » ($f(x) = \text{Sin}[x]$), où $T_n(x) \to f(x)$ pour tout x lorsque $n \to \infty$. Ils peuvent voir ainsi l'utilité du concept du polynôme de Taylor :



L'exemple « moins idéal » ($f(x) = \text{ArcTan}[x]$), qui apparaît à la question suivante, encourage une reconnaissance aussi des limites du théorème : si $n \to \infty$, alors $T_n(x) \to f(x)$, mais parfois sur un intervalle restreint des x autour de a. Les images générées par les étudiants pour répondre à la quatrième question (voir, p.e., l'image ci-après) peuvent leur illustrer le concept crucial de rayon de convergence. Au cinquième exercice, les étudiants peuvent apprendre du fait que le rayon de convergence n'est pas toujours symétrique autour de a. En faisant les explorations permettant d'y arriver, ils peuvent aussi s'approprier le rôle de la constante a dans l'énoncé du théorème (la constante a n'est pas nécessairement zéro). Après avoir vécu des exemples compliqués, les dernières deux questions cherchent à rappeler aux étudiants que le polynôme de Taylor est en fait un polynôme : si $f(x)$ est un polynôme, alors $T_n(x) = f(x)$

pour tout n plus ou égal au degré de $f(x)$. Ceci étant dit, la question 7 peut montrer aux étudiants que lorsque n est moins que le degré d'un polynôme $f(x)$, le polynôme de Taylor correspondant peut inclure des termes qui ne se trouvent pas dans l'expression de $f(x)$.



Notons que cette activité peut aussi faire connaître aux étudiants de nouvelles façons d'utiliser le logiciel choisi, ainsi contribuant à leur genèse instrumentale de cet outil. Par exemple, si l'activité est complétée à l'aide de *Mathematica*, les étudiants peuvent apprendre comment utiliser les fonctions préprogrammées comme « Sum », « Derivative » et « Plot ». En étudiant l'utilisation de telles fonctions, ils peuvent également apprendre à employer (plus efficacement) la documentation du logiciel ou d'autres ressources en ligne. Et s'ils sont invités à discuter de leurs approches, en groupe ou avec toute la classe, ils peuvent aussi développer leurs compétences de programmation plus générales ; par exemple, ont-ils pensé à programmer le polynôme de Taylor comme une fonction qui dépend de f, x, a et n ?

Finalement, en faisant cette activité, les étudiants peuvent être mis en contact avec des processus qui ressemblent à ceux vécus par des mathématiciens professionnels : notamment, la création, la vérification et l'utilisation d'un outil d'exploration, la réalisation, l'interprétation et l'explication des observations, le développement et l'évaluation des conjectures et même la présentation et la discussion des résultats.