6-17-2020 1:00 PM

Investigating an Approach to Integrating Computational Thinking into an Undergraduate Calculus Course

Erin Clements, *The University of Western Ontario*

Supervisor: Gadanidis, George, *The University of Western Ontario*
Co-Supervisor: Namukasa, Immaculate, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Education

© Erin Clements 2020

Abstract

Computational thinking can be conceptualized as patterns of thinking which align with certain fundamental computer science processes. While this algorithmic way of thinking has always been integral to computer science, it has recently gained momentum as a valuable approach to problem solving in a wide variety of contexts. Education researchers highlight the potential of computational thinking to transform, enrich, and revitalize teaching and learning experiences, by providing a systematic framework for analysis and enabling powerful computational tools to be incorporated to further enhance problem-solving activities. Research suggests that in order to maximize the affordances of computational thinking, it should be integrated into all subjects, from primary to tertiary, in meaningful and subject-specific ways. However, due to persistent theoretical and practical barriers, comprehensive integration of computational thinking into school and university curricula has not yet been achieved. One particularly strong obstacle identified in the literature is the lack of practical resources detailing how to effectively incorporate computational thinking into subjects beyond computer science. Using a case study research design with over 1000 participants, my project investigated an approach to integrating computational thinking into a first-year calculus course at McMaster University. Students engaged in computational thinking by working on computer coding activities developed to complement the mathematical content taught in the course. Following each set of activities, students responded to prompts designed to determine: (1) how students' conceptual understanding of calculus concepts changes in response to working on problem-solving and mathematical modelling activities which incorporate computational thinking, and (2) how students' learning experiences are transformed when they explore calculus concepts, ideas and techniques using computational tools and models. A qualitative content analysis of these responses revealed that exploring calculus concepts with code modified students' perceptions of mathematics, enhanced their mathematical learning experiences, and offered unique coding affordances. Further analyzing the data using a literacy framework helped situate the results of this study within the broader context of a computational literacy. This research augments the ongoing project, *Computational Thinking in Mathematics Education*, by providing insights and rich feedback on an approach to designing and integrating coding activities into a tertiary mathematics curriculum.

Keywords

Computational thinking, tertiary mathematics, computational literacy, calculus, authentic (real-life) applications, coding, modelling.

Summary for Lay Audience

Computational thinking describes a collection of thinking patterns and problem-solving strategies which are common in computer science. Recently, education researchers have suggested that this algorithmic way of thinking has the potential to transform, enrich, and revitalize teaching and learning experiences in a wide variety of disciplines. Despite this recognition, persistent theoretical and practical barriers have prevented its widespread integration into school and university curricula. The current study investigated an approach to integrating computer coding activities, which encourage and support computational thinking, into a first-year calculus course at McMaster University. An analysis of students' feedback revealed that exploring calculus concepts with computer code modified their perceptions of mathematics as a discipline, enhanced their mathematical learning experiences, and presented unique opportunities to interact with mathematical concepts in novel ways. This study provides fresh insights into an approach to designing and integrating coding activities into a tertiary mathematics curriculum, augmenting on-going research projects in this area.

Dedication

I would like to dedicate this work to my mom, for demonstrating an unparalleled work ethic throughout my life, and teaching me the importance of resilience.

And to my husband, Chris, and daughter, Sophie, for always believing in me, putting things into perspective, and motivating me to dream big.

Acknowledgments

First and foremost, I would like to thank my supervisor, George Gadanidis, for providing me with the opportunity to expand my perspective of mathematics education and inspiring me to (finally!) learn how to code. Thank you for accommodating my busy life schedule, providing consistent feedback, and supporting me through the most difficult times in my academic career. And to my co-supervisor, Immaculate, thank you for providing valuable feedback and encouragement along this pathway—it was greatly appreciated!

I would like to thank the Mathematics and Statistics Department at McMaster University for their continuous support of my research activities on campus and for always accommodating my schedule at Western. This research would not have been possible without the full support and openness to innovation in mathematics education present in our department.

I would also like to give a huge thanks to the fall 2018 Math 1LS3 students, who provided rich, valuable feedback throughout this entire research project. And on a larger scale, thank you to all of my students, who bring joy, laughter, and sometimes tears into our teaching and learning experiences and consistently motivate me to improve mathematics education. Teaching a course is never the same twice and I will always value the fresh perspective and enthusiasm my students bring into my professional life.

And finally, my most heartfelt thank you goes to my dear friend Miroslav Lovric. Your encouragement, patience, and understanding have been immeasurable (in any space) throughout the 20+ years we've known each other. Thank you for your consistent faith in me, and your unwavering support throughout the better part of my life.

Table of Contents

List of Figures

List of Appendices

Chapter 1

1 Introduction

The present study investigates an approach to integrating computational thinking into an undergraduate calculus course at McMaster University. In particular, using a case study research design, this project focuses on how students' conceptual understanding of calculus concepts changes when they engage in computational thinking activities, and how their learning experience transforms when these activities are integrated into their mathematical explorations, problem solving and modelling. This study has the potential to contribute to research aimed at exploring initiatives in, and affordances of, computational thinking in undergraduate mathematics education.

1.1 Computational Thinking

Computational thinking can be characterized as a systematic way of thinking about, exploring, analyzing, and—if feasible—formulating solutions to a wide range of problems. It involves abstracting key features of a problem and reformulating it so that a solution can be computed as an algorithm (i.e., automated). While not inextricably linked to computers, as the adjective "computational" might suggest, computational thinking encompasses a collection of thinking patterns and problem-solving strategies which align with certain computer programming processes and techniques.

The usefulness of computational thinking has been widely recognized in the field of computer science, where the ideas are directly applied to programming; however, more recently, attention has turned toward the potential of computational thinking to enhance logical reasoning skills and enrich problem-solving experiences in a diverse range of contexts. To date, computational thinking has facilitated and innovated research in nearly all disciplines, where it has been used to generate new knowledge and investigate questions in ways inconceivable before its implementation (Bundy, 2007). Yadav, Mayfield, Zhou, Hambrusch, and Korb (2014) note the pervasiveness of computational thinking in the present era and assert that the principles of computing—in particular, computational thinking—"influence every aspect of our lives, from shopping with loyalty cards to conducting scientific research" (p. 5:1).

Research in education has identified a vast set of competencies—applicable across subjects, contexts, and disciplines—that could be acquired by actively engaging with computational thinking at all levels, from kindergarten to university, and beyond. Furthermore, computational thinking affordances have the potential to influence, enrich, and revitalize learners' experiences in unique and transformative ways. Despite this recognition, comprehensive integration of computational thinking into school and university curricula has not yet been realized. One particularly strong barrier, frequently identified in the literature, is the lack of practical approaches, along with research-based evidence, which are needed to effectively incorporate computational thinking into classroom instruction, particularly outside of computer science courses.

1.2 Key Terms

I defined computational thinking in the previous section. Here I include its brief characterization, to contrast with other key terms used throughout this thesis.

Computational thinking – a collection of thinking patterns, tools, and strategies which parallel fundamental computer programming concepts and processes.

Computational modelling – adopting computational thinking strategies to reason about, explore, analyze and solve problems, which involves reformulating the problem so that it may be remediated with computer code.

Computer coding/computer programming – using a programming language (e.g., Python 3) and a computer-based coding environment (e.g., Jupyter notebook) to represent key features of a problem, and designing structures and algorithms (e.g., loops, tables of data, matrices, etc.) needed to analyze and solve it.

Mathematical modelling – reformulating an application (real-world, authentic) problem using mathematical objects (e.g., functions and equations) and procedures (e.g., integration, numeric algorithms) to provide answers, results and insights about the application.

Literacy – a broadly adopted system of representing, analyzing, and communicating ideas.

Computational thinking involves two fundamental processes: abstraction and automation. In order to begin exploring a mathematical problem or concept using code, the problem must first be abstracted (i.e., broken down into its basic elements, with key features, structures, and relationships identified) and reformulated (e.g., from an algebraic form into a numerical representation) so that it can be remediated with a computational representation (code). The next step requires devising an algorithm suitable for solving the problem, and then creating a computational model for this algorithm (i.e., generating the code required to carry out the algorithm) to automate a solution. This process, referred to as computational modelling, illustrates the connection between computational thinking and computer coding.

1.3 Purpose of the Current Study

My doctoral research project strives to address a gap in the literature in integrating computational thinking into the existing undergraduate mathematics curriculum. The evidence for this claim is based on my extensive literature search, and confirmed by several researchers in mathematics education that I consulted. There is strong motivation to do so—I argue that computational thinking provides an essential new approach that facilitates mathematical modelling, an important component of mathematics education that connects mathematics to authentic, real-world problems. Anecdotal, as well as research-documented, evidence suggests that students struggle to apply mathematics they have learned (or have been exposed to) not just to real-world applications (Stillman, 2015) but also to subsequent courses in mathematics and elsewhere. (For instance, Pepper et al. (2012) document students' challenges in applying mathematics concepts they were supposed to know in a course on electricity and magnetism.) Blum and Ferri (2009) refer to the analyses carried out by the Programme for International Student Assessment (PISA) Mathematics Expert Group, which attribute students' difficulties with mathematical modelling to the “inherent cognitive complexity” (p. 48) of the modelling tasks. Tall et al. (1987) show that exploring advanced, complex concepts (such as integration and mathematical modelling which involves abstract mathematical topics) from a multitude of perspectives (including computer-aided exploration) helps students overcome cognitive difficulties and facilitates understanding.

The goal of this project was to develop, implement, and analyze a practical approach to integrating computational thinking, in the form of computational modelling and computer coding, into teaching and learning activities in the undergraduate applied calculus course I teach—Math 1LS3[1]—without removing any mathematical content from the course syllabus. While there have been successful attempts at developing courses designed to teach mathematics and coding simultaneously (for example, the Mathematics Integrated with Computers and Applications (MICA) program at Brock University), the large-scale intervention implemented in this research project, that is, integrating coding into a preexisting calculus course (with a class size of over 1000 students), has not been attempted before at a Canadian university (and possibly beyond). Additionally, this project presents a tangible approach to generating course-specific computational thinking activities and resources and documents the effectiveness of this intervention in a large, diverse undergraduate calculus class.

Rather than be discouraged by the lack of practical resources identified in the literature, I viewed this deficiency as an unprecedented opportunity to bring innovation into my classroom from a course-specific perspective, and to create computational modelling activities that were meaningful and relevant for life sciences students. I strived to design valuable experiences for my students, where they had the opportunity to experience maximum educational benefits from an engagement with computational thinking. It was important to me that the approach I implemented would be feasible in a large classroom

[1] Math 1LS3 is a first-year, undergraduate applied calculus course designed for life science majors. It is a prerequisite course for the program as well as many upper year courses, and so there is very little flexibility in the topic coverage.

and would provide equitable, inclusive opportunities for a diverse group of students with varying backgrounds, mathematical experiences, and learning abilities.

1.4 Research Questions

My inquiry was guided by the following research questions:

1. How does students' conceptual understanding of calculus concepts change in response to working on problem-solving and mathematical modelling activities which incorporate computational thinking?
2. How are students' learning experiences transformed when they explore calculus concepts, ideas and techniques using computational tools and models?

1.5 Research Design

To investigate my research questions, I used a case study research design and collected data from students enrolled in Math 1LS3 at McMaster University during the fall semester of 2018. Taken by over 1500 students every year, this foundational course teaches basic concepts of differential and integral calculus, with a heavy emphasis on applications in the fields of life and health sciences.

To integrate computational thinking into Math 1LS3, I created a series of coding activities to complement and enrich the topics studied in our course. These activities invited students to explore calculus concepts and solve problems using computational models, thus engaging them in computational thinking. They were organized into four computer labs, which corresponded to the main themes in the course: mathematical models, limits and derivatives, differential equations and integrals, and discrete-time dynamical systems. Additionally, coding was almost effortlessly incorporated into our lectures and coursework, as numerous topics and ideas naturally required a computational approach (e.g., Euler's Method for solving differential equations, Riemann sums, and discrete-time dynamical systems).

At the end of each computer lab, students were invited to reflect on and share their experiences with certain specific aspects of the coding activities (for example, "Describe how your perspective on a mathematical idea changed as a result of engaging with the coding activities in Lab 2”). I collected and analyzed the open-ended survey responses from all students who gave their consent and reported on these findings.

1.6 Methodology and Theoretical Framework

Upon receiving each data set, I conducted a qualitative content analysis, facilitated by NVivo 12—a qualitative data analysis computer software package. In NVivo, I coded each response according to key terms, main topics, and general sentiment. I used annotations throughout the coding process to record my observations, thoughts, and questions, and I kept a detailed research journal to provide an auditable trail of my analytical process. Once all data had been coded, I organized the nodes into categories (higher-level nodes) and wrote detailed memos for each category, including illustrative examples from my raw data. These memos were clustered into three overarching themes and form the basis of my Results chapter.

I then considered Andrea diSessa’s (2018) theoretical framework for a computational literacy and analyzed my results according to his five literacy principles. This framework helped me evaluate my results against diSessa’s literacy criteria, which, in turn, allowed me to establish a correspondence between my findings and diSessa’s principles. This correspondence is presented in my Analysis chapter.

1.7 Significance of the Study

This research contributes to the ongoing, Social Sciences and Humanities Research Council (SSHRC)-funded research project *Computational Thinking in Mathematics Education* (http://ctmath.ca/), from pre-school to undergraduate mathematics, and in mathematics teacher education. In particular, it investigates a large-scale implementation (involving approximately 1000 students) of a specific approach to integrating computational thinking (in the form of coding activities) into a tertiary level mathematics course. The affordances of computational thinking were explored from a mathematical teaching and learning perspective (rather than from a computer science perspective); that is, the activities were designed to complement and enrich the mathematical ideas and teaching and learning ecology.

1.8 Limitations

My extensive literature search (confirmed by anecdotal evidence from several researchers in mathematics education I consulted) found no existing research close to the current study (integration of coding into an existing first-year university math course); therefore, I am not able to directly compare my data and results with other research efforts. Thus, conducting this study once is a limitation, which could be remedied by repeating the study in different semesters, possibly in different courses, or in other universities.

All results are based on students self-reporting their experiences and opinions, which is a subjective process. Unfortunately, due to several reasons, it was not possible (nor was it the objective here) for the researcher (myself) to observe individual students working on the labs (however, that is an idea for future research). Since self-reported data cannot typically be independently verified (Rukwaru, 2015), we assumed that students were sufficiently self-aware and cognizant of their learning and experiences, and that their comments accurately reflected their experiences, thoughts and opinions. My reported findings reflect *common* themes, that is, the categories in my Results chapter emerged multiple times independently, that is, as responses from many students.

The sample of students surveyed for this research project is biased (the majority of students taking Math 1LS3 are life sciences majors), and thus directly transferring computer labs to other departments and universities might produce somewhat different results. So while my results are not necessarily universal, given the large sample size, I can confidently say that they are definitely representative of life sciences students.

1.9 Summary

Computational thinking is gaining recognition as a versatile analytical approach, which can innovate and transform problem-solving activities in a wide variety of contexts. In particular, unique affordances of computational thinking can be employed to influence, impact, and reorganize the field of education in novel ways. The advantages seem to be particularly beneficial when computational thinking is effectively integrated into existing subjects; however, the guidance on implementation outside of computer science (e.g., what are the best, evidence-supported teaching strategies) is insufficient. My research project aims to address this gap by investigating an approach to integrating computational thinking into an applied undergraduate calculus course designed for life sciences majors.

Chapter 2

2 Review of the Literature

I begin this chapter by outlining a working definition of computational thinking, and illustrating the two fundamental processes involved—abstraction and automation—with a concrete mathematical example. I then summarize the body of literature surrounding computational thinking, paying particular attention to its current role in education, and the unique opportunities it affords mathematical problem solving. I conclude this chapter by discussing certain theoretical and practical barriers, which currently impede its comprehensive integration into school and university curricula.

2.1 Computational Thinking

Computational thinking encompasses a collection of problem-solving strategies that derive from fundamental computer science principles, processes and techniques (Curzon, Black, Meagher, & McOwan, 2009). These core concepts and capabilities include—but are not limited to—data representation and abstraction, problem decomposition and reduction, algorithmic and recursive thinking, automation, and simulation (Wing, 2006; Barr & Stephenson, 2011). The Royal Society describes computational thinking as "the process of recognizing aspects of computation in the world that surrounds us, and applying tools and techniques from computer science to understand and reason about both natural and artificial systems and processes'' (Furber, 2012, p. 29). Stephen Wolfram (2016) adds, "its [computational thinking] intellectual core is about formulating things with enough clarity, and in a systematic enough way, that one can tell a computer how to do them" (para. 6).

Yadav et al. (2014) posit that "the prominent features of computational thinking revolve around abstraction and automation, indicating the ability to dissect problems, abstract the high-level rules, and use technology to automate the problem-solving process" (p. 5:1). Wing (2010) also emphasizes the centrality of these concepts and explains, "The most important and high level thought process in computational thinking is the abstraction process. Abstraction is used in defining patterns, generalizing from instances and parameterization” (p. 1). Wing (2008) continues, “Computing is the automation of our abstractions” (p. 3718).

Abstraction involves reformulating a problem so that it can be computed as an algorithm, that is, as “a series of steps that control some abstract machine or computational model without requiring human judgment” (Denning, 2017, p. 33). This process begins with logically deconstructing the problem into smaller, more manageable sub-problems, therefore reducing its complexity. An important part of the abstraction process is deciding which features of the problem should be accentuated and which details are to be viewed as insignificant. Wing (2008) asserts that this important decision-making process (which is a cornerstone of mathematical or statistical modelling) underlies computational thinking. Next, variables and parameters must be chosen and adequately abstracted, and features of the solution (e.g., patterns) need to be anticipated, so that adequate structures (e.g., matrices and databases) and techniques (e.g., algorithms) can be utilized.

Automation requires “systematically devising an algorithm suitable for solving” the problem and its sub-problems (Mohaghegh & McCauley, 2016, p. 1524). This stage is guided by, and depends on, the computational model employed to approach the particular problem. Aho (2012) explains, “Mathematical abstractions called models of computation are at the heart of computation and computational thinking” (p. 834). He asserts that “finding or devising appropriate models of computation to formulate problems is a central and often nontrivial part of computational thinking” (p. 833), especially when computational thinking is being used to investigate problems in domains for which the classical models from computer science (such as the Turing model of sequential computation) may neither be appropriate, nor adequate, nor sufficient. diSessa (2018) refers to these considerations as “the representation effect”—exploring what can adequately and usefully be reformulated and represented in a computational form.

The final stage of computational thinking involves an execution of the computer code (i.e., running algorithms) that yields a solution to the problem, followed by an evaluation. In this final stage, an in-depth analysis of both the product (solution) and the process (automation and abstraction) is conducted. The initial stages of computational thinking (abstraction and automation) can be performed with or without technology; however, the final stage (analysis) requires the use of a computer or other suitable technology.

2.1.1 Example: Intermediate Value Theorem

As an illustration, consider the Intermediate Value Theorem, which is a common topic in a first-year calculus course. Formally, it is stated in the following way (Stewart, 2012, page 125):

If $f(x)$ is a continuous function defined on a closed interval $[a, b]$ and N is a number between $f(a)$ and $f(b)$, then there is a number c in $[a, b]$ such that $f(c) = N$.

In other words, the Intermediate Value Theorem states that a continuous function $f(x)$ on a closed, finite interval $[a, b]$ attains all values between $f(a)$ and $f(b)$; see Figure 1.

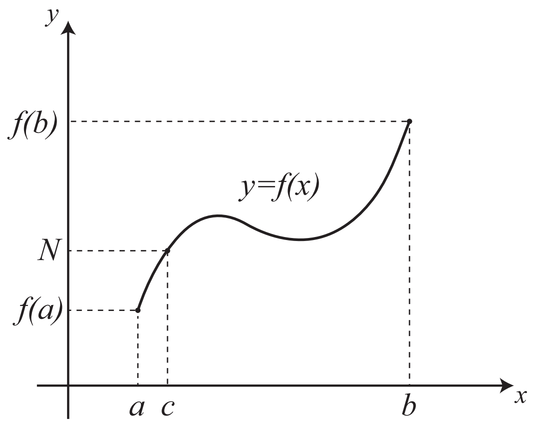


Figure 1: Illustration of the Intermediate Value Theorem.

A typical, first-year undergraduate calculus problem might require students to use the Intermediate Value Theorem to show that the equation $e^x = 5x + 10$ has a solution for $3 \leq x \leq 4$. To prove this, we let $f(x) = e^x - 5x - 10$ and $N = 0$. We calculate that $f(3) \approx -4.9$ and $f(4) \approx 24.6$. Since $f(x)$ is continuous on $[3, 4]$, the Intermediate Value Theorem guarantees that there is a number c in the interval $[3, 4]$ such that $f(c) = 0$; see Figure 2. Note that we did not find the value of c—in fact, this is impossible to solve for algebraically—we just proved that such a value of c exists.

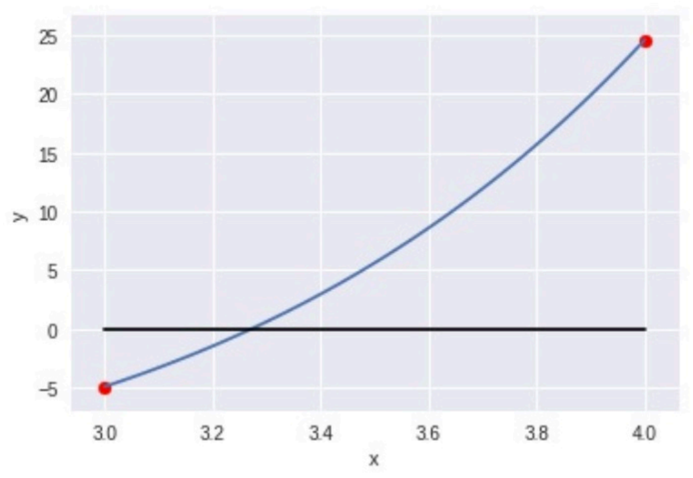
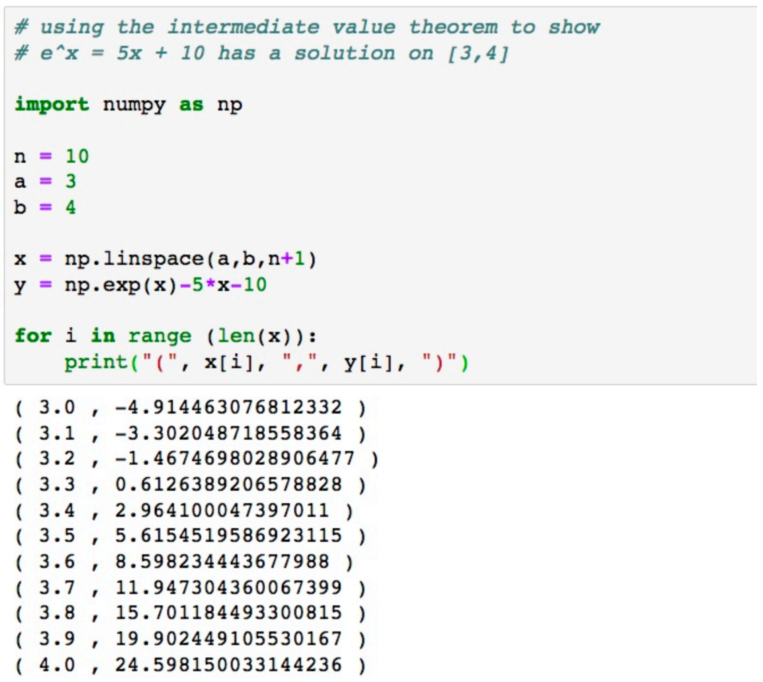


Figure 2: The equation $e^x = 5x + 10$ has a solution between 3 and 4.

To reformulate this problem so that it can be computed as an algorithm, we begin by discretizing the interval $[3, 4]$ into a finite number of equally spaced x-values and then computing the corresponding values of $f(x)$. That is, we reformulate this continuous function as a discrete set of values (points). We then need to inspect the list of $f(x)$ values to see if there is a value c for which $f(c) = 0$. Alternatively, if we observe a sign change between two consecutive values $f(x_i)$ and $f(x_{i+1})$, then we know that there must be a value c in $[x_i,\ x_{i+1}]$ such that $f(c) = 0$. Observe the output in Figure 3; in this case, the sign change occurs between $x_i = 3.2$ and $x_{i+1} = 3.3$. This process accomplishes two tasks: first, it proves that there is a solution on the interval $[3, 4]$; and second, it narrows down the interval on which there is a solution from $[3, 4]$ to $[x_i,\ x_{i+1}]$. Recognizing the value in the second observation, we can keep repeating this process (which demands an automated algorithm!) and narrowing down the interval until we have a solution c as close as desired to the actual value.

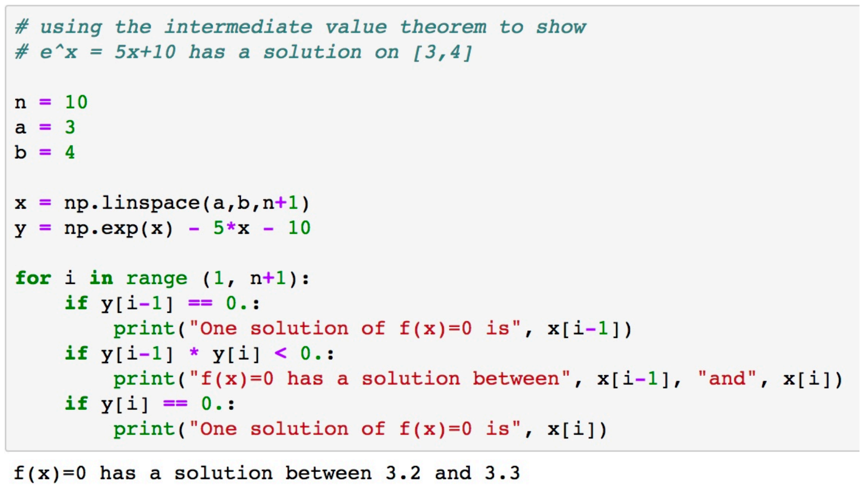


```python
# using the intermediate value theorem to show
# e^x = 5x + 10 has a solution on [3,4]

import numpy as np

n = 10
a = 3
b = 4

x = np.linspace(a,b,n+1)
y = np.exp(x)-5*x-10

for i in range (len(x)):
    print("(", x[i], ",", y[i], ")")
```

```
( 3.0 , -4.914463076812332 )
( 3.1 , -3.302048718558364 )
( 3.2 , -1.4674698028906477 )
( 3.3 , 0.6126389206578828 )
( 3.4 , 2.964100047397011 )
( 3.5 , 5.6154519586923115 )
( 3.6 , 8.598234443677988 )
( 3.7 , 11.947304360067399 )
( 3.8 , 15.701184493300815 )
( 3.9 , 19.902449105530167 )
( 4.0 , 24.598150033144236 )
```

Figure 3: Abstraction of the Intermediate Value Theorem.

Now that the problem has been adequately abstracted, how can we automate the solution process so that we do not have to visually inspect long lists of $f(x)$ values? A rather simple idea is to create code that will detect either a zero or a sign change within the list of $f(x)$ values. The code in Figure 4 accomplishes this task.

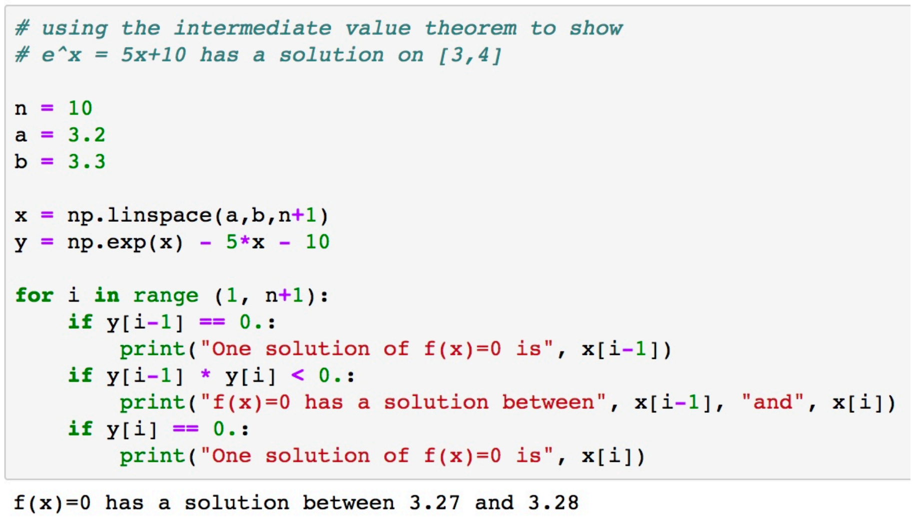


```python
# using the intermediate value theorem to show
# e^x = 5x+10 has a solution on [3,4]

n = 10
a = 3
b = 4

x = np.linspace(a,b,n+1)
y = np.exp(x) - 5*x - 10

for i in range (1, n+1):
    if y[i-1] == 0.:
        print("One solution of f(x)=0 is", x[i-1])
    if y[i-1] * y[i] < 0.:
        print("f(x)=0 has a solution between", x[i-1], "and", x[i])
    if y[i] == 0.:
        print("One solution of f(x)=0 is", x[i])
```

```
f(x)=0 has a solution between 3.2 and 3.3
```

Figure 4: Automation of the Intermediate Value Theorem.

After generating the list of $f(x)$ values, the code scans through to look for either a zero (thus, actually identifying the precise value c) or a negative product between two consecutive y-values (which indicates that one was positive and the other was negative). Once a solution is located, the interval can be further refined by adjusting the parameters a and b, as seen in Figure 5.



```
# using the intermediate value theorem to show
# e^x = 5x+10 has a solution on [3,4]

n = 10
a = 3.2
b = 3.3

x = np.linspace(a,b,n+1)
y = np.exp(x) - 5*x - 10

for i in range (1, n+1):
    if y[i-1] == 0.:
        print("One solution of f(x)=0 is", x[i-1])
    if y[i-1] * y[i] < 0.:
        print("f(x)=0 has a solution between", x[i-1], "and", x[i])
    if y[i] == 0.:
        print("One solution of f(x)=0 is", x[i])
```

```
f(x)=0 has a solution between 3.27 and 3.28
```

Figure 5: Automation of the Intermediate Value Theorem. Refining the interval on which the equation has a solution.

This reformulation has limitations, of course. For example, if we begin with too coarse of a refinement, we may miss a solution (or solutions) altogether, or we might find one, but fail to detect other solutions. Combining this algorithm with a complementary geometric representation and analysis of the problem helps to avoid situations such as this one.

2.2 Value of Computational Thinking

The value of computational thinking has been widely recognized in the field of computer science from the very beginning—that is, with the first attempts at writing code to solve specific problems. More recently, attention has turned toward the potential of computational thinking to enhance thinking and problem solving in a broad array of contexts, while enabling technology to be effectively incorporated to generate a solution. According to Wolfram (2016), “computational thinking provides a framework that makes things more transparent and easier to understand" (para. 78). Moreover, the various competencies developed through an engagement with computational thinking—such as self-motivation to explore, experiment and hypothesize, development of intuition about variables, relations and quantities (i.e., quantitative literacy), logical reasoning, abstraction, and critical reflection—extend far beyond the scope of computer science (King, Hillel, & Artigue, 2001; Marshall & Buteau, 2014).

Weintrop et al. (2016) report, "in the last 20 years, nearly every field related to science and mathematics has seen the growth of a computational counterpart" (p. 128). The authors describe how computational methods have been employed in novel ways to explore and study stochastic and nonlinear problems, many of which were previously inaccessible, or unsolvable. These innovative applications of computational thinking have expanded the range of phenomena that can be investigated using mathematical models and simulations to include systems which generate chaotic behavior, and complex dynamical systems in general.

Bundy (2007) further extends the scope of computational thinking when he asserts, "computational thinking is influencing research in nearly all disciplines, both in the sciences and the humanities" (p. 1). For example, in biology, computational thinking enabled the accelerated sequencing of the human genome and provided opportunities to model complex biological processes, such as the cell cycle and protein folding (Wing, 2008). In the humanities, computational thinking has been used to analyze data from a vast number of literary sources to illustrate relationships between the prevalence of certain words and themes as functions of time, location, political situation, or other variables. For instance, in the analysis of a Shakespearean play, computational thinking can be used to create and analyze a social network of characters and interactions, facilitating an in-depth study of the intricacies and relationships within the play (Wolfram, 2016). Computational thinking has allowed researchers to model complex interactions between geological processes, thus obtaining a more comprehensive understanding of the historic, as well as future, dynamics of our planet (Bundy, 2007). This improved understanding can help geologists "understand, predict and influence the mechanisms involved in climate change" (Bundy, 2007, p. 2). Adding to philosophical discussions, Donald Hoffman combined aspects of computational thinking with evolutionary game theory to outline a compelling proof that what we believe is reality is actually just our perception (Gefter, 2016).

Computational thinking extends into commerce, where market research is continuously conducted based on the analysis of a user's Internet browsing history, resulting in personally customized advertisements and product endorsements. In politics, hundreds of thousands of sources of information are effectively and efficiently scanned daily (making use of powerful algorithms) to provide up-to-date information on relevant issues, political views, and voter support.

Wing (2008) believes that while, so far, computational thinking has been successfully employed to tackle relatively simple (i.e., solvable) problems involving data mining and simulations, in the future, far more complex uses of computational thinking will help us to discover deeper meanings and understandings hidden in the patterns that can be extracted from the huge quantities of data that is generated and collected daily.

In light of the expanding range of computational thinking, Wing (2006) has advocated "To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (p. 33). Grover and Pea (2018) explain that computational thinking "is now recognized as a foundational competency for being an informed citizen and being successful in STEM [science, technology, engineering and mathematics] work, one that also bears the potential as a means for creative problem solving and innovating in all other disciplines" (p. 20). Weintrop et al. (2016) echo this and assert that the changing landscape of STEM fields presents a challenge to "bring current educational efforts in line with the increasingly computational nature of modern science and mathematics" (p. 127).

2.3 Computational Thinking Affordances in Mathematics Education

Computational thinking provides a framework for a systematic investigation of a problem and it affords the use of technology to extend both teaching and learning beyond present constraints. The Computer Science Teachers Association explains that “the study of computational thinking enables all students to better conceptualize, analyze, and solve complex problems by selecting and applying appropriate strategies and tools, both virtually and in the real world’’ (Computer Science Teachers Association, 2011, p. 9). While computational thinking does not require the use of technology, computer programming (that is, creating and running computer code) reinforces computational thinking in multiple ways (Wing, 2008) and can be incorporated to further enrich problem-solving activities.

Mathematics and computational thinking share a natural connection in that the processes operating when one works on a computational problem (such as experimenting, logical reasoning and algorithmic thinking) align with those employed in mathematical problem solving. As well, since computational thinking “complements and combines mathematical and engineering thinking” (Wing, 2008, p. 35), it is reasonable to presume that the tools and strategies developed for computational thinking can be employed to facilitate learning and enhance problem-solving skills in mathematics. In fact, Sanford and Naidu (2016) assert that computational thinking “can and does augment, facilitate, and expand the realm of thinking, logic, and mathematics” (p. 24).

Computational thinking provides new perspectives and insights into a problem and allows for innovative approaches, such as experimentation, animation and simulation, to be explored (Pesonen & Malvela, 2000). As Sanford and Naidu (2016) attest, “A thorough study of any problem becomes easy with the aid of a computer and students can be encouraged to innovate and investigate other situations” (p. 24). For instance, the usual analysis of algebraic solutions of an equation can be augmented by a computer-driven experiment in which solutions to an entire set of equations, similar to the original one, are generated. This approach may deepen students’ understanding of the link between the features of an equation (i.e., its coefficients) and the corresponding properties of its solutions.

Anecdotal evidence suggests that many students find linear transformations challenging, in part due to the fact that common approaches (pencil and paper and calculator) fail to produce a sufficient number and variety of examples. Through animation and simulation (i.e., by seeing how various objects change under a large number of linear transformations), one can get a good “feel” for this abstract algebraic concept. Students often have difficulties making sense of the algebraic manipulations required to solve an equation; however, after “playing” with equations on a computer (for instance, by moving a slider which controls a coefficient and watching how solutions react) students have the opportunity to develop a more solid understanding of what the solutions of a given equation are supposed to look like. This new understanding could, in turn, guide them through (previously incomprehensible) algebraic steps towards calculating a solution.

When students explore a mathematical concept using a code, they are giving abstractions a “tangible feel” (Gadanidis, 2015). For instance, the abstract concept of the domain of a function becomes “tangible” when students realize that the computer returns an error message to their request to calculate the square root of a negative number. This feedback diagnostic is the “tangible” realization of the abstract fact that the domain of the square root does not include negative numbers. As this example demonstrates, the objectification of abstractions (Hazzan, 1999) could help students understand and internalize challenging ideas and concepts.

Euler’s method provides another powerful example. This numerical method uses successive tangent line approximations to estimate the solution to a first-order initial value problem. By hand, or with a calculator, we can only compute a small number of tangent lines to approximate a curve, and are forced to extrapolate the properties of a solution, which remains abstract, and “hidden” from view. Remediating this situation with a computational approach allows us to decrease the step size as desired and to subsequently increase the number of tangent line approximations to generate a “tangible” approximation to the entire solution curve (which we can also visualize by graphing).

According to Yadav et al. (2014), “Computational thinking has the potential to advance students’ problem-solving skills and abilities significantly as they begin to think in new ways” (p. 5:2). As well, using multiple avenues to investigate a problem (e.g., algebraic, visual and dynamic) not only facilitates in-depth learning and promotes an active engagement with the concepts, but also helps to accommodate students with a wide spectrum of abilities and learning styles.

Wing (2006) explains that computational thinking tools enable us to reduce the complexity of any given task by "reformulating a seemingly difficult problem into one we know how to solve" (p. 33). The process of deconstructing a complex problem into manageable sub-problems—performed during the abstraction phase of computational thinking —is a useful problem-solving strategy in a myriad of situations.

Dynamic modelling—a significant affordance of computational thinking—allows students to explore relationships between the key features of a problem in novel ways. This interactive approach provides several significant advantages over traditional ways of learning. For instance, modelling complex interactions between two species sharing the same ecosystem is virtually impossible without employing computer code. As well, within a coding environment, students are able to interact with the variables and parameters and observe an immediate response (feedback) to their input. As Burton (1999) explains, when students have the ability to actively control and manipulate these components, both their learning and attitude towards mathematics are enhanced. The immediate feedback provided when executing code, often given in the form of a dynamic visual or an animation, helps students to form concrete representations of abstract mathematical concepts.

Rich visualizations of mathematical relationships generated through coding activities have the potential to further enhance mathematical understanding. Recent evidence from neuroscience research suggests, "our mathematical thinking is grounded in visual processing" (Boaler, Chen, Williams, & Montserrat, 2016, p. 2). The authors state that "when students learn through visual approaches, mathematics changes for them, and they are given access to deep and new understandings" (p. 1). Computational thinking, through coding, experimentation (for instance by using simulations), and dynamic visuals (such as graphs and diagrams) provides numerous opportunities for students to explore mathematics visually, thus further promoting active engagement and enhancing mathematical development.

Digital environments, which facilitate computational thinking, can offer significant educational advantages. Gadanidis, Hughes, Miniti and White (2016) report that a number of existing computer programming (coding) environments provide “low floor, high ceiling, wide walls” experiences to users. The “low floor, high ceiling” property offers multiple entry points and maximum engagement opportunities for a wide range of students with diverse abilities, background knowledge and experiences. The “wide walls” feature allows the coding environment to be used for “many different types of projects so people with many different interests and learning styles can all become engaged” (Resnick et al., 2009, p. 63). Furthermore, since various coding environments—such as Scratch or Jupyter notebook— as well as a multitude of interactive lessons and tutorials are available online free of charge, computational thinking can be “successfully taught to a very wide range of people, regardless of their economic resources” (Wolfram, 2016).

In a computational thinking environment, students create tools required to solve a problem, instead of using existing tools (Mohaghegh & McCauley, 2016). This creative process encourages students to become active producers, instead of passively using prepackaged content and approaches to solving problems. As Gadanidis et al. (2016) explain, “When students write computer code to model a relationship, they are in control. They can write the code in ways that personally make sense, and they can deviate from the specific task to explore related problems or extensions” (p. 15). This sense of agency and control has the potential to motivate students, build their confidence, and stimulate independent learning. Seymour Papert—who pioneered the use of computers to teach children mathematics—asserts, “I am convinced that the best learning takes place when the learner takes charge” (Papert, 1993, p. 25).

Czerkawski and Lyman (2015) and Denning (2017) are careful to note the important distinction between computing and computational thinking and assert that simply using a computer to facilitate problem solving does not necessarily imply that computational thinking is being employed. While engaging in computer programming activities facilitates the development of computational thinking skills, the cognitive benefits are most significant—and transferable outside of computer science—when students acquire a deep understanding of the underlying principles of computing. As Mohaghegh and McCauley (2016) point out, "What is necessary is an effective integration of the 'tool' with the concepts" (p. 1528); consequently, the process, as well as the product (i.e., the solution), should be studied and thoroughly understood. They further explain that "deeper understanding of computational problem solving is more valuable than exploring the surface of tools in this area without realising their full potential" (p. 1527). Wing (2008) echoes this point when she makes the comparison of using a computer without understanding the principles of computing to using a calculator without understanding how to do the calculations. As well, Grover and Pea (2013) report that "current computational tools vary in their effectiveness in allowing for engagement with the various component elements of computational thinking" (p. 41).

2.4 Brief History of Computational Thinking and Its Role in Education

Computer scientist Peter Denning (2017) defines computational thinking as "the habits of mind developed from designing programs, software packages, and computations performed by machines" (p. 33). Thus, the development of the concept of computational thinking is, not surprisingly, closely tied to the advancement of computers and programming languages, which were created as a means of communicating with computers.

In the 1960's, Alan Perlis, a computer scientist and professor at Yale University, described programming as "an exploration of process" (Guzdial, 2008, p. 25), which he argued is relevant to every student, irrespective of their field of study. Working on the development of programming languages and anticipating their potential and importance, Perlis proposed that all university students should learn to program. He claimed that the logical and algorithmic thinking skills (later referred to as "computational thinking skills") attained through writing and analyzing computer code would, with suitable practice, transfer into areas outside computer science.

Advancements in computer technology, in particular time-sharing, allowed a large number of students to interact with a computer, thus making computer-assisted learning a real, tangible goal. Subsequently, computer-based instructional materials were created for all subjects and for all levels of education (Molnar, n.d.).

In the early seventies, based on Jean Piaget's work and his own learning theory (constructionism), Seymour Papert and his collaborators created a visual programming language called Logo. Its purpose was to help children improve their thinking and problem-solving abilities and to facilitate learning mathematics through coding in an environment which promoted play and experimentation. Papert's unique platform provided a "low floor, high ceiling" experience, that is, students were able to engage with a problem with very little background knowledge in programming, however, Logo had the potential to explore complex, high-level problems in mathematics. In his influential book *Mindstorms*, Papert postulated that "computer presence could contribute to mental processes not only instrumentally but in more essential, conceptual ways, influencing how people think even when they are far removed from physical contact with a computer'' (Papert, 1980, p. 4). Reflecting upon the impact of new technologies on the way children learn, he further suggested that, "learning to use computers can change the way they learn everything else" (p. 8).

In spite of major advances in the final quarter of the twentieth century (e.g., Internet and laptop computers) learning with computers did not find its way into every classroom. There were computer labs and computer-programming courses, but the ideas that the early pioneers put forward, namely of integrating computers and computational thinking into school and university curricula, did not materialize.

At the start of the 21st century, MIT physics professor Andrea diSessa studied the concept of "computational literacy," a potentially new form of literacy that has the power to modify the way people think and learn. diSessa separated the "cognitive" aspect of computational thinking from the "material" aspect and suggested that computing can be used to explore fields other than computer science (Grover & Pea, 2013). "I view computation as, potentially, providing a new, deep, and profoundly influential literacy—

computational literacy—that will impact all STEM disciplines at their very core, but most especially in terms of learning" (diSessa, 2018, p. 4). Continuing the line of work started by Papert, diSessa's research aims "to bring computational ideas, indeed, programming, to the wider population for general intellectual purposes" (pp. 19-20).

Despite its early emergence and presence in computer science and related literature, computational thinking was not given serious attention by the majority of educators until Jeannette Wing published her influential article, *Computational Thinking* (Wing, 2006). In it, she argued that computational thinking is a powerful and fundamental cognitive skill for everyone, and that children should develop computational thinking proficiencies alongside other important core analytical abilities, such as reading, writing, and arithmetic.

In the second decade of this century, a growing number of educators have acknowledged the benefits of integrating computational thinking into school and university curricula, as the proficiency in computational thinking and related skills seems to be the best way to prepare students for the challenges that the future will bring.

2.5 Current State of Computational Thinking in School and University Curricula

Today, we witness computational thinking gaining recognition as "an essential skill for those who would be our future inventors, innovators, and shapers of culture and public discourse" (Pearson, 2009, p. 42). The National Council for Research (2010) refers to computational thinking as "a cognitive skill that an average person is expected to possess" (as cited in Yadav et. al, 2014, p. 5:2). Consequently, students of all ages, irrespective of the discipline they study, are expected to develop competencies in various aspects of computational thinking in order to meet the demands of an increasingly digital world.

Recent research suggests that the advantages of computational thinking are maximized when computational thinking is introduced—in adequate form—to students at a young age, and effectively integrated into all subjects, providing a universal approach to problem solving (Sanford & Naidu, 2016; Yadav, Zhou, Mayfield, Hambrusch, & Korb, 2011). As Yadav et al. (2014) state, the “pervasiveness of computational-thinking concepts dictates the importance of exposing students to such notions early in their school years and helping them to become conscious about when and how to apply these ideas” (5:2). Weintrop et al. (2016) argue that integrating computational thinking into existing subject areas—rather than teaching it as a standalone course—provides multiple benefits. This integration provides meaningful, authentic contexts in which to study computational thinking; it addresses the practical issues of supplying proficient teachers and resources; it allows for computational thinking activities and practices to reach the widest possible audience; and—especially in mathematics and sciences—it brings “education more in line with current professional practices in these fields” (p. 143).

Several countries, including England, Israel, Russia, New Zealand, U.S., Australia and South Africa have incorporated computational thinking into their K-12 curricula, often within computer science or computer programming courses (Grover & Pea, 2013). A recent document published by the European Commission investigates major trends in integration of computational thinking with compulsory education, outlines approaches to teaching, learning and assessment, and discusses teacher training in computational thinking (Bocconi, Chioccariello, Dettori, Ferrari, & Engelhardt, 2016). The Next Generation Science Standards published by U.S. educators includes computational thinking as an important learning objective and outlines activities and suggestions to help teachers promote this skill within the classroom (Sneider, Stephenson, Schafer, & Flick, 2014). The document *Computing in the National Curriculum: A Guide for Primary Teachers* (Berry, 2013) written for teachers in the U.K. aims to “demystify the programme of study” (p. 3) of computing in primary schools. “It will enable teachers to get to grips with the new requirements quickly and to build on current practice. It includes help for schools with planning and gives guidance on how best to develop teachers’ skills” (p. 3).

Additionally, building competencies in computational thinking has become a requirement in many undergraduate university programs, as computational thinking has become essential for the development and learning of all STEM disciplines (Henderson, Cortina, Hazzan, & Wing, 2007), as well as those outside of STEM (Czerkawski & Lyman, 2015). By arguing that computational thinking is critical thinking, Kules (2016) builds arguments for connecting computational thinking to university discourse. In his thesis, Kolodziej (2017) investigates important elements of this connection, including domain expertise, interdisciplinary collaboration, and attitudes towards curricular initiatives. Swaid (2015) contributes to "efforts to establish computational thinking as a *universally applicable attitude* that is meshed within STEM conversations, education and curricula" (p. 3657).

A number of online initiatives aim to provide both experiences and education in computational thinking that are free and accessible to learners of all ages and abilities. For example, MIT's Scratch provides a visual-programming environment, which uses coding "blocks" to create and run computer code (https://scratch.mit.edu/). Graphical programming environments, such as Scratch, "allow early experiences to focus on designing and creating, avoiding issues of programming syntax" (Grover & Pea, 2013, p. 40). Google produced online lessons and exercises in computational thinking for both educators and students through Project Bloks, commonly referred to as "Google Bloks" (https://projectbloks.withgoogle.com/). A less technology-dependent complement to these efforts can be found on the webpage, *CS Unplugged* (http://csunplugged.org/), which aims to teach the fundamentals of computer science without the use of computers. The collection of activities available on the site provides multiple opportunities to engage with computational thinking in various interactive ways and is suitable for all age levels and abilities.

While encouraging, these efforts are still insufficient, as most consist of working on isolated curriculum objectives (e.g., within computer science or programming courses), rather than focusing on genuine integration with other subjects (Grover & Pea, 2013). Gadanidis et al. (2016) concur, and allege that at the K-12 level, computational thinking is not yet "integrated with curriculum to enrich existing subject areas" (p. 1). Czerkawski and Lyman (2015) report that the response to the call for a pervasive computational thinking presence in higher education is "scattered" (p. 58) and note that although there have been many localized "clusters of cross-disciplinary interest" (p. 58) at integrating computational thinking into undergraduate curricula, “There is not yet a coherent cross-institutional movement to incorporate computational thinking as a fundamental skill-set, outside of computer science and a few STEM disciplines” (p. 58). According to Grover and Pea (2013), “although there is broad acknowledgement that computing pervades all aspects of the global economy, its place as a mandatory part of the school curriculum is far from secure” (p. 40).

Additionally, diSessa (2018) notes that many recent coding initiatives claim to include computational thinking, however, most only teach technical programming skills. Denning (2017) reminds us that using computational tools does not automatically imply that one is engaged in computational thinking.

2.6 Barriers to Integrating Computational Thinking into School and University Curricula

Presently, the major obstacles preventing widespread integration of computational thinking into K-16 curricula revolve around certain theoretical issues, general expertise, teacher education and teaching practice, as well as beliefs and attitudes of university faculty and instructors toward significant curricular changes. As Barr and Stephenson (2011) report, “The process of increasing student exposure to computational thinking in K-12 is complex, requiring systemic change, teacher engagement, and development of significant resources” (p. 48). Czerkawski and Lyman (2015) note that in higher education, sustained interdisciplinary interest, collaborations and outreach are essential in the pursuit of extending computational thinking beyond computer science courses.

The absence of a precise definition of computational thinking is a theoretical issue which frequently arises in the literature and is often cited as posing a significant barrier to the widespread propagation of computational thinking in education (Grover & Pea, 2013; Czerkawski & Lyman, 2015). As Aho (2012) explains, “the term computation means different things to different people depending on the kinds of computational systems they are studying and the kinds of problems they are investigating” (p. 832). Denning (2017) believes that the definition of computational thinking was intentionally designed to be vague in order to increase the perception of its expansive applicability outside of computer science. While defining computational thinking broadly avoids associating it with a particular discipline, Denning suggests that the definition has been widened beyond a practical boundary, thus losing its usefulness.

The lack of a clear definition is believed to contribute to confusion and misperceptions about computational thinking, resulting in less than ideal attitudes from stakeholders regarding the resources and efforts allocated to widening the integration of computational thinking into the K-16 curriculum. For example, a common misconception is that computational thinking reduces to thinking like a computer. Wing (2006) explains that computational thinking is simply an efficient, systematic, and analytical way of thinking, and asserts that "computational thinking is a way humans solve problems; it is not trying to get humans to think like computers" (p. 35). In fact, computational thinking is not as artificial to human thinking patterns as its terminology might suggest, nor should it be automatically associated with computer technology. The results from a study by Lewandowski et al. (2010) suggested that people without formal programming experience had an innate, yet underdeveloped, ability to reason correctly about certain computing principles, such as concurrency. Furthermore, Berland and Lee (2011) observed that students engaged in a strategic and collaborative (non-digital) board game demonstrated complex computational thinking practices—such as "conditional logic, distributed processing, debugging, simulation, and algorithm building" (p. 60)—continuously throughout the game (as cited in Czerkawski & Lyman, 2015). These findings suggest that humans naturally possess the cognitive foundations for computational thinking and that computational thinking skills can be further developed with adequate support and practice.

While the technical terms automation and abstraction might seem exclusively linked to computer science, Barr and Stephenson (2011) highlight multiple ways that these (and other fundamental computational thinking concepts and competencies) manifest themselves in various other disciplines. For example, the use of a simile or metaphor illustrates how abstraction might be employed in the language arts, whereas building a model of a physical entity—such as a molecule or cell—demonstrates the use of abstraction in the physical sciences. Using computational tools to efficiently handle certain routine problems, such as using a word processor in the language arts or Geometer's Sketchpad in mathematics, provides examples of how automation arises in contexts outside of computer science. Wing (2006) also explains how various aspects of computational thinking are involved in our day-to-day lives—for example, planning and executing our morning routine to get from bed to work or school involves multiple problem-solving strategies which align with those found in computing. While these practical and diverse examples help to illustrate the pervasiveness of computational thinking, Denning (2017) suggests that we need to be careful and restrictive in drawing parallels between algorithms (as conceptualized in the context of computational thinking) and routines that we "execute" in our daily lives. He reminds us that "an algorithm is not any sequence of steps, but a series of steps that control some abstract machine or computational model without requiring human judgment" (p. 33).

Several educational researchers have criticized the overzealous assertions made by computational thinking advocates, arguing that their claims are ambitious, overreaching and empirically unsubstantiated. In particular, claims of the universal value of computational thinking and the extent to which it can positively impact activities in all fields lack empirical support. Denning (2017) criticizes claims that "computational thinking enhances general cognitive skills that will transfer to other domains where they will manifest as superior problem-solving skills" (p. 37) since the universal value of computational thinking is, as of yet, empirically unsubstantiated. He concludes that what the current literature reveals at most is that "computational thinking primarily benefits people who design computations" (p. 37). diSessa (2018) expresses similar concerns over Wing's grand claim of universally applicable skills acquired through engaging in computational thinking. He cites historical research efforts that attempted to discover and develop "higher order thinking skills," that is, cognitive skills thought to be universally beneficial in a multitude of domains and contexts. The general consensus from these studies was that there is little or no evidence of the existence of "domain general skills" (p. 22). diSessa concludes that "Problem solving does not seem to be critically powerful, even in a single discipline let alone transformative across disciplines" (p. 22). Moreover, he reminds us that all representational systems have "distinctive and critical strengths, but also limitations and blind spots" (p. 7); that is, what can be represented with a computational model and significantly accomplished will vary greatly between different domains and within different situations. diSessa suggests a more modest claim on the potential of computational thinking may be more appropriate, at least until further research provides more concrete evidence of its “ubiquitous” power.

At the K-12 level, parents, students, and teachers have questioned the relevance of teaching computational thinking to students who show no interest in computer science or related disciplines. Hemmendinger (2010) emphasizes that introducing computational thinking into all subject areas should not be perceived as an attempt to train all students to become computer scientists, but rather “to teach them how to think like an economist, a physicist, an artist, and to understand how to use computation to solve their problems, to create, and to discover new questions that can fruitfully be explored” (as cited in Yadav et al., 2014, 5:2). Viewing computational thinking as part of a much larger scale achievement—computational literacy—diSessa (2018) reminds us not to be discouraged by initial unfavourable attitudes towards change and notes that the emergence of any new literacy is a long and complex social process in which “initial resistance and long periods of incubation are undoubtedly the norm” (p. 15). As more efforts to incorporate computational thinking come from within specific disciplines, and activities are thoughtfully designed to integrate computational thinking in authentic ways, students (and parents) may perceive its inclusion as a natural development within the discipline, rather than an external force driven by computer science objectives. This would likely lead to improved attitudes as relevance and alignment with practices in the field are realized.

Epistemological concerns have been raised in response to the idea that computational thinking should be embedded into all subjects as a universal approach to problem solving. Many non-science faculties reject positivist notions classically associated with the natural sciences in favour of interpretivist or constructivist paradigms and consequently “avoid analytical techniques that may be perceived as reductionist” (Czerkawski & Lyman, 2015, p. 62). Since computational thinking is still strongly associated with computer science, there might exist an assumption that it is limited by the same restrictions associated with computing. For example, the foundational element of computer science is the Turing machine—a closed, finite model of sequential computation. Problems considered solvable using this classical model of computation are limited to those that can be reformulated in a meaningful way to be Turing-machine computable. While this restriction narrowed the scope of problems that could be computed, the emergence of new models of computation, such as natural computing models and concurrent interactive computing models (Dodig-Crnkovic, 2011), have significantly expanded the type of problems that can be profitably computed. As well, Soh et al. (2009) hypothesize that interdisciplinary training for computer scientists "may encourage the development of computational thinking methods suitable to the 'open-ended' issues studied in the humanities and fine arts" (as cited in Czerkawski & Lyman, 2015, p. 62). Czerkawski and Lyman (2015) discuss how computational thinking can be viewed as a human-computer collaboration, that is, a reciprocal relationship that "both expands the range of human creativity by incorporating computational thinking and expands computational thinking by promoting the development of new models of interactive computing" (p. 62). As these innovative, open models of computation become more mainstream, and the value that computational explorations can bring to less conventional, non-traditional problems is adequately illustrated, resistance to remediating problems with computation will likely decrease.

A more practical issue to consider is that there is generally a "lack of orientation toward domain-specific adaptation" (diSessa, 2018, p. 27) in regards to selecting, adapting, and transferring salient computational thinking skills outside of a computer science context. This is vastly apparent in education, where integrating computational thinking into various subject fields in the K-12 curriculum is under-investigated from a teachers' perspective (Grover & Pea, 2013). While the general consensus is that computational thinking should be embedded into existing subjects rather than taught in an isolated context, exactly what this integration is supposed to look like in practice is still vague (Lye & Koh, 2014). As a result, there are minimal resources available which provide practical teaching strategies, exercises, and assessment principles necessary for its full implementation (Grover & Pea, 2013). Yadav et al. (2014) also note this deficiency and report, "there is very little research on how teachers could be prepared to incorporate computational thinking ideas in their own teaching" (p. 5:13). For example, a teacher might be aware of the advantages of computational thinking, but might not know how to incorporate computational thinking into a particular content or lesson. As Yadav et al. (2014) note, “most of the current efforts to educate teachers about computational thinking have been limited to computer science teachers” (5:3). As well, the lack of appropriate teaching materials (textbooks, manuals, study sheets, templates for activities, and so on), combined with inadequate teacher preparation results in teachers’ reluctance to engage their students with computational thinking in a meaningful capacity. Czerkawski and Lyman (2015) report that this deficiency is even greater at the university level and note, “practical research on teaching computational thinking skills continues largely to take place within computer science and the science, technology, engineering and math (STEM) fields” (p. 60). They observe that “Outside of computer science and the STEM fields, the difference between applying computational thinking methods derived from computer science and simple application of computers to problems within a discipline (‘data crunching’) is either less well-understood or simply elided” (p. 58).

2.7 Recent Theoretical and Practical Advancements

Barr and Stephenson (2011) report on a multiphase project that was launched in 2009 by the Computer Science Teachers Association and the International Society for Technology in Education with the goal of “developing an operational definition of computational thinking for K-12 along with suitable resources for policy and curricular change” (p. 49). They emphasize the core concepts or capabilities of computational thinking which are both common with, and transferable to, other disciplines and provide practical examples demonstrating how computational thinking concepts can be applied in various core subject areas.

By analyzing routines that were viewed as important for both students and the interdisciplinary practices within mathematics, Weintrop et al. (2016) developed a taxonomy for computational thinking in mathematics and sciences, which provides “a sharper definition that is distinct from computer science, yet still grounded in authentic, meaningful computational practices that are essential for students to master” (p. 128). Their work presents a significant contribution toward bringing computational thinking into classrooms relatively quickly and effectively, that can “serve as a resource to address ‘what’ and ‘how’ questions that accompany the creation of new educational materials” (p. 129).

Additionally, Yadav et al. (2011) demonstrated that when pre-service teachers participated in a computational thinking training module, their attitude toward and understanding of computational thinking improved and they reported to be more likely to implement computational thinking in their classrooms. A later study by Yadav et al. (2014) found that pre-service teachers presented with relevant information in computational thinking demonstrated a more comprehensive and nuanced understanding of computational thinking overall and were able to articulate its centrality in other disciplines and suggest innovative ways it could be integrated into classroom teaching for a wide variety of subjects. This research suggests that computational thinking principles, practices, and related activities should be incorporated into pre-service teachers’ coursework regardless of their content specialization, as a first step towards the goal of integrating computational thinking into all subject areas.

In post-secondary education, there have been a moderate number of cross-disciplinary efforts to integrate computational thinking into curriculum; however, as Czerkawski and Lyman (2015) note, so far these endeavors have been localized in scope. For example, at the University of Nebraska-Lincoln (UNL), Soh et al. (2009) created a framework for non-computer science majors of “multiple pathways through a series of computer science courses that were specialized according to the students’ main areas of study” (as cited in Czerkawski & Lyman, 2015, p. 63). In 2001, Brock University began offering a four-year mathematics program—Mathematics Integrated with Computers and Applications, or MICA—that combines a foundational tertiary mathematics education with computing and information technology. This unique program focuses on solving authentic, complex real-world problems by effectively integrating mathematics and computation. Additionally, numerous universities offer introductory courses on the principles of computing that do not involve any programming. For example, the Department of Mathematics and Statistics at McMaster University offers an undergraduate course (*Topics in Logic*) open to students in all faculties, which focuses exclusively on the theory of computation and does not involve the use of computers.

Czerkawski and Lyman (2015) conclude their review of the state of computational thinking in higher education by identifying areas where further research is needed in order to make computational thinking a more pervasive, cross-disciplinary skill. They suggest that part of these efforts should aim to "Establish methods and strategies as well as examples and cases for teaching computational thinking in various non-technical disciplines, especially the social sciences, humanities and education" (p. 64).

2.8 Summary

Computational thinking has always played a fundamental role in computer science; however, more recently, it has gained recognition for its potential to innovate, transform, and enrich educational experiences by providing a systematic framework for analyzing a problem, and enabling powerful computational tools to be incorporated to further enhance problem-solving activities. While some researchers (Denning, 2017; diSessa, 2018) point out that ambitious claims of the ubiquitous benefits of computational thinking to teaching and learning lack empirical evidence, a body of literature suggests (and education researchers concur) that such claims are worth a further, more thorough investigation. The goal of the current study is to contribute meaningful data and analysis to this research pursuit, in the context of the integration of computational thinking into tertiary mathematics.

Chapter 3

3 Methodology

In this chapter, I describe my research design, the participants and setting, the particular research intervention, and data collection. I discuss my theoretical and analytical frameworks, and explain (with specific examples) how I conducted my content analysis. I then describe how I used diSessa's (2018) literacy framework to help situate my results within the broader context of a computational literacy. I conclude this chapter with a discussion of the trustworthiness of the study, and the measures I have taken to ensure that this research is credible, transferable, confirmable, and dependable.

3.1 Research Questions

My current research project investigates an approach to integrating computational thinking into an undergraduate applied calculus course that I have implemented at McMaster University. This was accomplished by supplementing mathematical problem-solving activities with appropriate, carefully designed computer coding activities, which were incorporated into lectures to explore mathematical concepts and illustrate coding techniques, and organized into a set of computer labs to be used as an assessment component. I collected and analyzed students' responses to a series of questions posed at the end of each lab, which invited them to reflect on their experiences with the mathematical coding activities.

In order to determine how integrating computational thinking into my students' learning environment affected their understandings and experiences, I formulated the following two research questions:

1. How does students' conceptual understanding of calculus concepts change in response to working on problem-solving and mathematical modelling activities which incorporate computational thinking?
2. How are students' learning experiences transformed when they explore calculus concepts, ideas and techniques using computational tools and models?

3.2 Research Design

To answer my research questions, I used a case study research design, which Stake (2005) advises is most appropriate for collecting descriptions of teaching and learning experiences within a bounded system (i.e., the thoughts and actions of participants in a specific education setting, for example, students working on mathematics problem-solving exercises using a computational tool, such as a laptop or a tablet, in a specific course). This approach helped me to understand the main features of such a system as it functioned under natural conditions (Stake 2005; Yin, 2009). As well, Yin (2009) suggests that a case study research design is most appropriate for investigating "how" and "why" questions, such as the research questions posed in this study. My analysis is qualitative in nature, following the established practice of in-depth studies of classroom-based learning and case studies in general (Stake, 2005), and uses qualitative content analysis to identify key themes in the teaching and learning experiences.

3.3 Theoretical Framework

In my education research, I align myself with a social constructivist epistemology, identifying with Vygotsky (1978) in the belief that knowledge is constructed in interactions with others. In this context, I expand Vygotsky's concept of "others" to include technology, that is, I believe that knowledge is constructed when humans interact with computer technology. Like Borba and Villareal (2005), I believe that students' mathematical thinking and knowledge can fundamentally change by interacting with mathematics using technology. Ontologically, I adopt a relativist viewpoint, since I view reality as being comprised of "local and specific co-constructed realities" (Lincoln, Lynham, & Guba, 2011, p. 100).

3.4 Participants and Setting

I conducted this research at McMaster University in the fall of 2018, with students enrolled in Math 1LS3, a first-year undergraduate calculus course designed for life sciences majors. This foundational course teaches the basic concepts of differential and integral calculus, with a heavy emphasis on applications in the fields of life and health sciences. During the fall 2018 semester, there were approximately 1020 students enrolled in Math 1LS3 and the students were assigned to one of three lecture sections, each taught by a different instructor. All sections covered the same topics in approximately the same order (the course coordinator maintained a common course webpage for students and instructors to follow) and students in all sections were given identical assignments, computer labs, tests, and final exam.

During lectures, instructors introduce standard calculus topics (such as functions, limits, derivatives, integrals, discrete-time dynamical systems) and then apply these concepts to investigate mathematical models within biological contexts. To demonstrate relevance, models are taken from journals in life and health sciences, and deal with contemporary situations, such as drug abuse and spread of viral diseases. Students remark that they recognize the value of this approach, however, they frequently admit (to myself, other instructors, and on their course evaluations) to having a great deal of difficulty working with these complex models.

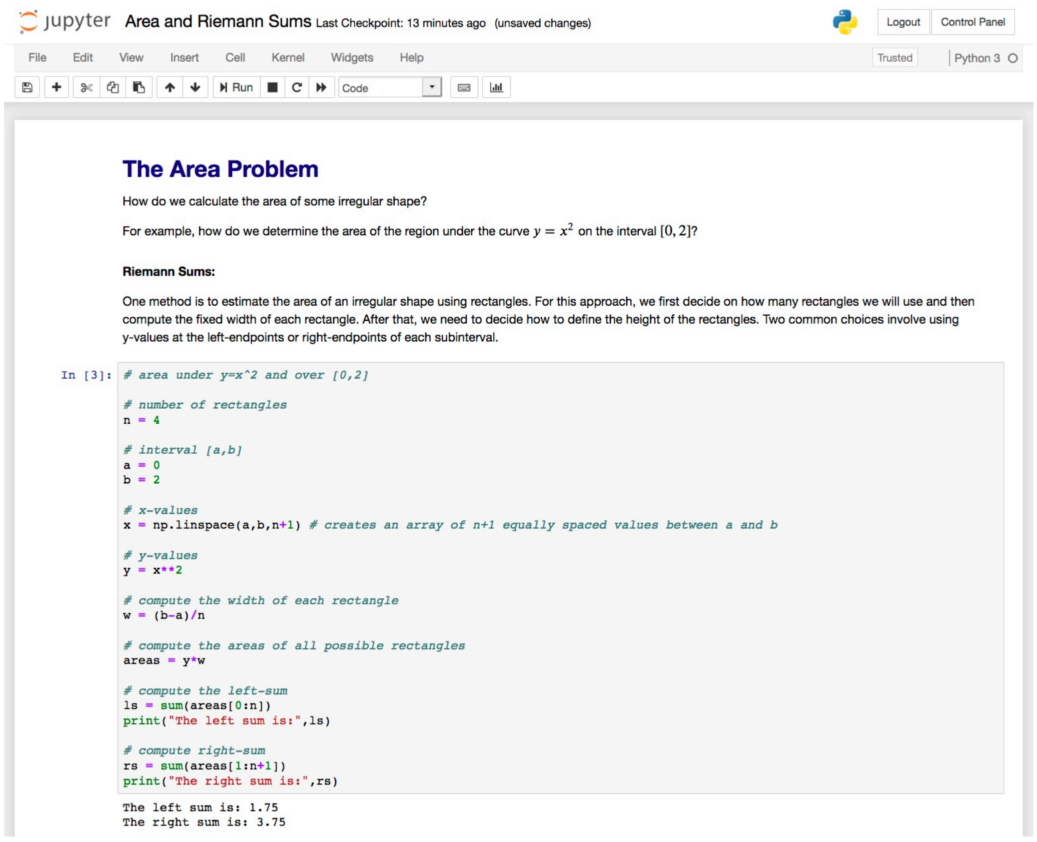
Throughout the course, students work on a series of assignments to reinforce the material being presented in lectures. For practical reasons, these assignments have been designed so that students are able to complete them without the use of computer technology (as well, many mathematics courses at McMaster do not allow the use of calculators on tests and final exams). In reality, applying calculus techniques to complex, real-life models without the use of computational thinking tools often requires long, tedious and time-consuming calculations. In order to be adapted to actual teaching practices, these modelling tasks are necessarily over-simplified to generate special cases, which can be investigated algebraically, or by using a hand-held calculator. Unfortunately, this often reduces the perceived value of the mathematical application being considered. For example, the calculations required to approximate the solutions to a system of differential equations—such as the SEIR-model, used to study the spread of the EBOLA virus during the recent epidemic (Althaus, 2014)—using Euler's method with a large enough number of iterations to obtain meaningful results, would be unwieldy without computer technology. In traditional approaches, students are asked to approximate the solution to a single differential equation by applying Euler's method for a maximum of four steps (which is straightforward to compute), thus obtaining a superficial, uninteresting, and, from the application point of view, useless answer.

3.5 Intervention

In collaboration with the course coordinator, I developed a collection of coding activities to complement the mathematical material studied in Math 1LS3 and to facilitate a deeper, enriched exploration of the course content in more authentic contexts. These activities invited students to explore calculus concepts and solve problems using computational models, thus engaging them in computational thinking. The activities were organized into four computer labs (see Appendix C for Lab 3), which corresponded to the main themes in the course: mathematical models, limits and derivatives, differential equations and integrals, and discrete-time dynamical systems.

All lab content—theory, examples, explanations, data sets, pictures, and sample code—was organized into a Jupyter notebook[2] file, which students were able to access and download from the course webpage 10 days before each lab was due. Mindful that Math 1LS3 is not formally a combined mathematics and computer science course, we presented the coding activities as a numerical approach to the mathematical ideas and focused on directly applying the code, rather than teaching nuances of the programming language (Python 3). Brief explanations were provided as needed in the form of comments throughout the code (for example, x = np.linspace(0, 2, 4) #creates an array of four equally spaced x-values between 0 and 2 (see Figure 6)).

[2] A Jupyter notebook is a free integrated development environment, which supports HTML, Python, and R code (see Figure 6).



The Area Problem

How do we calculate the area of some irregular shape?

For example, how do we determine the area of the region under the curve $y = x^2$ on the interval $[0, 2]$?

Riemann Sums:

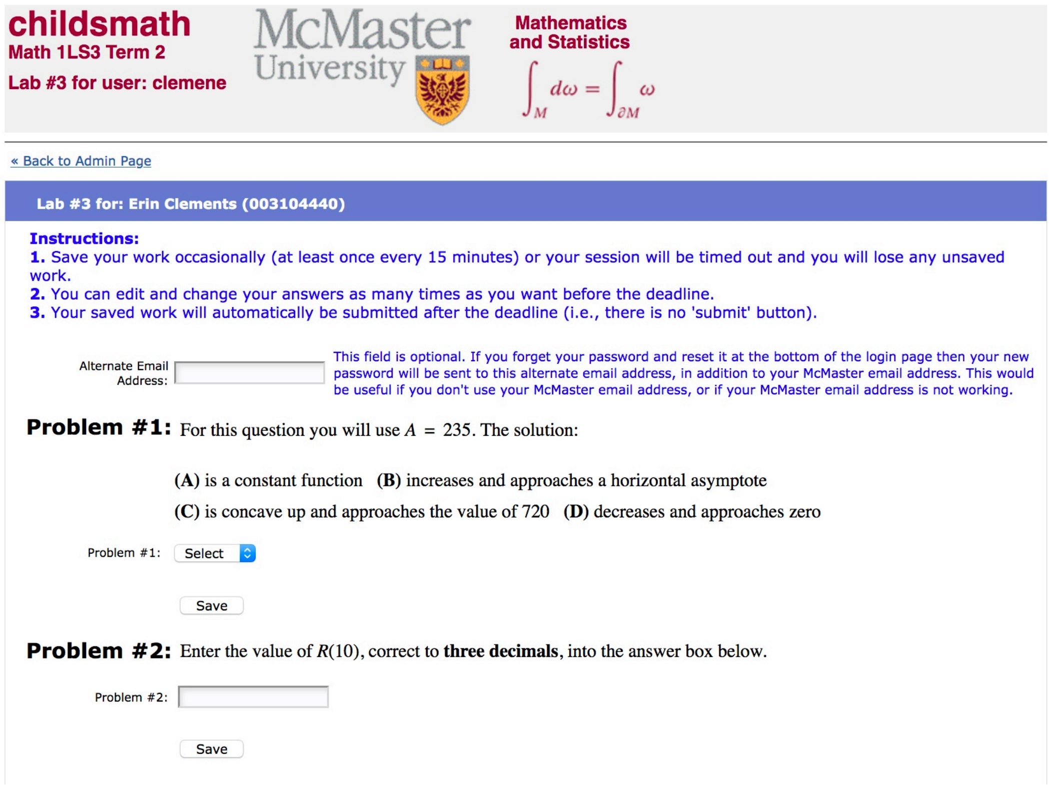
One method is to estimate the area of an irregular shape using rectangles. For this approach, we first decide on how many rectangles we will use and then compute the fixed width of each rectangle. After that, we need to decide how to define the height of the rectangles. Two common choices involve using y-values at the left-endpoints or right-endpoints of each subinterval.

```
In [3]: # area under y=x^2 and over [0,2]

        # number of rectangles
        n = 4

        # interval [a,b]
        a = 0
        b = 2

        # x-values
        x = np.linspace(a,b,n+1) # creates an array of n+1 equally spaced values between a and b

        # y-values
        y = x**2

        # compute the width of each rectangle
        w = (b-a)/n

        # compute the areas of all possible rectangles
        areas = y*w

        # compute the left-sum
        ls = sum(areas[0:n])
        print("The left sum is:",ls)

        # compute right-sum
        rs = sum(areas[1:n+1])
        print("The right sum is:",rs)
```

```
The left sum is: 1.75
The right sum is: 3.75
```

Figure 6: Screenshot of the Jupyter notebook integrated development environment.

Students were invited to explore and experiment with data, models, and algorithms by modifying the code directly in the online file. Throughout each lab, they were prompted to respond to a series of procedural and conceptual questions related to the coding activities, as well as a series of open-ended questions, which invited them to reflect on and evaluate their experiences with the coding activities and explain how these experiences affected their understanding of the mathematical content. Students submitted their responses electronically thorough *childsmath*—an internal survey and assessment tool created and maintained by a professor (Aaron Childs) in the Mathematics and Statistics Department, which is also used by several other Mathematics and Statistics courses at McMaster (see Figure 7). The first Jupyter notebook cell of each lab contained a link to *childsmath* and students would presumably toggle between the Jupyter notebook and *childsmath* windows as they completed each lab.


childsmath
Math 1LS3 Term 2
Lab #3 for user: clemene

Mathematics and Statistics

$$\int_M d\omega = \int_{\partial M} \omega$$

« Back to Admin Page

Lab #3 for: Erin Clements (003104440)

Instructions:
1. Save your work occasionally (at least once every 15 minutes) or your session will be timed out and you will lose any unsaved work.
2. You can edit and change your answers as many times as you want before the deadline.
3. Your saved work will automatically be submitted after the deadline (i.e., there is no 'submit' button).

Alternate Email Address:

This field is optional. If you forget your password and reset it at the bottom of the login page then your new password will be sent to this alternate email address, in addition to your McMaster email address. This would be useful if you don't use your McMaster email address, or if your McMaster email address is not working.

Problem #1: For this question you will use $A = 235$. The solution:

(A) is a constant function **(B)** increases and approaches a horizontal asymptote
(C) is concave up and approaches the value of 720 **(D)** decreases and approaches zero

Problem #1: Select

Save

Problem #2: Enter the value of $R(10)$, correct to **three decimals**, into the answer box below.

Problem #2:

Save

Figure 7: Screenshot of *childsmath* survey and assessment tool.

Math 1LS3 contained four term assessments, collectively worth 60% of students' final grade: test 1, test 2, test 3, and the fourth being the set of four computer labs. In computing final grades, only students' top three of these four assessments were used, each contributing 20% to the final grade. Thus, the completion of these computer labs had the potential to contribute 20% to students' overall grade.

Whenever appropriate, instructors incorporated coding activities into lectures to demonstrate various computational (numeric) approaches in mathematics. This helped to keep our explorations of applications meaningful and authentic, and also helped students develop the technical skills needed to complete the coding modules.

3.6 Data Collection

For each computer lab, the procedural and conceptual responses submitted through *childsmath* were automatically graded according to an answer key. The system generated an Excel spreadsheet for each lab, which contained all students' graded responses to the procedural and conceptual questions, as well as their text-based reflective responses to the open-ended questions. The spreadsheets were accessed and downloaded by our course coordinator, and then shared with our research assistant, Reihaneh Jamalifar.

Jamalifar read and graded each of the reflective responses, assigning one mark per question if there was a reasonably thoughtful response given, and zero marks if the field was left blank or if the response was deemed unacceptable.

Examples of acceptable (i.e., receiving one mark) responses:

> *I enjoyed the dynamic, interactive nature of the coding activities more than just solving problems on paper.*
>
> *The activities in this lab allowed me to experiment with different cases until I was able to fully understand the problem (and solution).*
>
> *I found the coding exercises within this lab to be too overwhelming without having explicit lessons on Python 3, and so the activities just confused me more.*

Examples of unacceptable (i.e., receiving zero marks) responses:

> *blahblahblah*
>
> *I am just writing something to fill the space to get a mark.*

Jamalifar calculated a "reflective response" grade for each student and submitted these grades to the course coordinator. She then sorted the reflective responses into two categories: students who consented to have their responses analyzed for research purposes, and those who did not. She removed all identifying data (e.g., names, student numbers, email addresses) and then exported the comments of students who gave their consent onto a USB drive. The reflective responses of students who did not give their consent were deleted.

We followed appropriate McMaster Research Ethics Board (MREB) and Western Research Ethics Board (WREB) guidelines and protocols, and obtained necessary approvals; see Appendix A for the Letter of Information. To maintain separation between my roles as (1) researcher and (2) instructor of one of the course sections, I did not have access to, and did not analyze, the data collected by Jamalifar until after the final course grades were submitted and approved.

3.7 Content Analysis

Following the completion of the fall 2018 semester, I accessed the data and conducted a qualitative content analysis. Hsieh and Shannon (2005) define a qualitative content analysis as "a research method for the subjective interpretation of the content of text data through the systematic classification process of identifying themes or patterns" (p. 1278). This inductive method is appropriate for interpreting the results of naturalistic inquiry, that is, an inquiry of a phenomenon researched in its natural setting (Hsieh & Shannon, 2005). As Krippendorff (2004) explains, "content analysis provides new insights, increases a researcher's understanding of particular phenomena, or informs practical actions" (p. 18).

I used the qualitative data analysis software package NVivo to facilitate my content analysis, and stored and organized my raw data, nodes, annotations, memos, pictures and research journal within this system. NVivo provided a detailed analytic framework, tutorials, guidance, suggestions, and examples for conducting a rigorous, systematic content analysis.

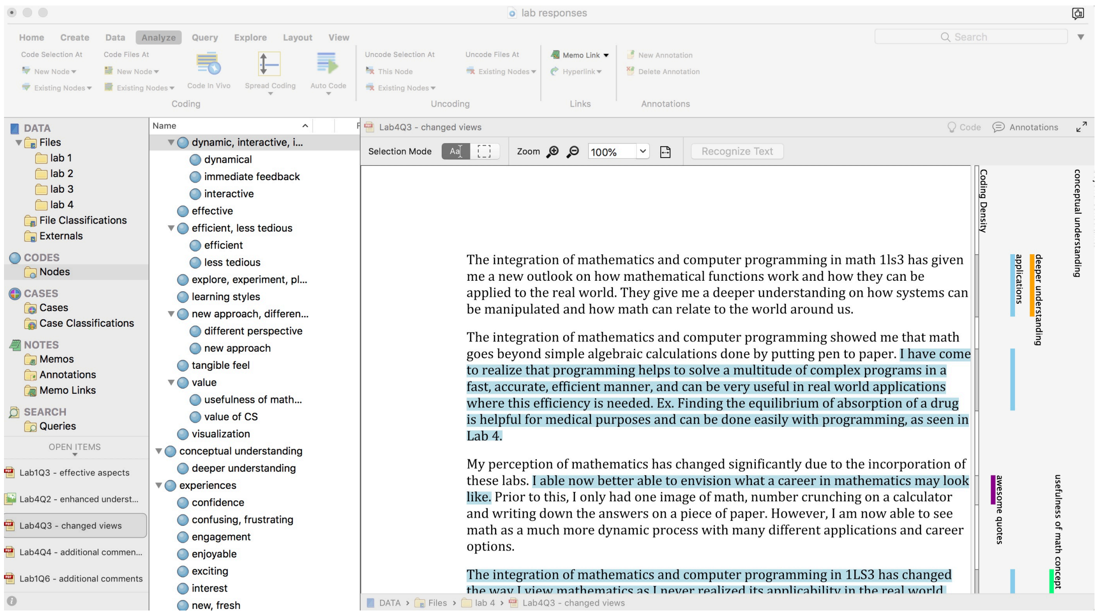
The first thing I set up within NVivo was a research journal to document the evolution of my project. I regularly reflected on my analytical process, and wrote detailed notes (Lincoln & Guba, 1985) of how I was "informed, redirected, and surprised by my data" (NVivo, n.d., Ways to get started with your project section) and eventually, how higher-level nodes, categories, and themes were discerned.

To begin my content analysis, I imported all consented open-ended survey responses from the computer labs into NVivo. I systematically read each comment to immerse myself in the data and to obtain a global, comprehensive sense of it (Hycner, 1985 as cited in Cohen, Manion, & Morrison, 2007; Tesch, 1990 as cited in Hsieh & Shannon, 2005). Concurrent with my reading, I identified key words and ideas, and recorded my "first impressions, thoughts, and initial analysis" (Hsieh & Shannon, 2005, p. 1279). This enabled me to generate an initial node structure, a process described by Mayring (2000) as "inductive category development" (Hsieh & Shannon, 2005, p. 1279). Using topic coding, I coded the data based on key words and used text search queries to find related comments in other sources (for example, other sets of responses). Using analytical coding, I considered how each particular comment related to my research questions. As Krippendorff (2004) explains, "research questions are the targets of the analyst's inferences from available texts" (p. 31); therefore, a persistent focus on my research questions was crucial throughout my analysis. I found that this more complex, refined approach required a thorough reading and reflection of the content in order for the data to be accurately coded. It was during this analytical coding that I found myself making extensive and frequent annotations, continuously comparing (Cohen, Manion, & Morrison, 2007) all comments coded at a particular node to ensure that they reflected the same sentiment.

For example, the student's response below was coded at four notes: "tangible feel," "visualization," "deeper understanding," and "new approach:"

> *… the computer lab helped* ***transform abstract concepts****, such as blood alcohol concentration, into* ***visual representations*** *through graphs. This helped* ***deepen the understanding*** *of the content as we could rely on* ***multiple learning approaches*** *rather than just conventional methods.*

I continued in this manner to code each comment at relevant nodes, creating new nodes as needed, until all comments from the four labs were coded. As Wilkinson and Birmingham (2003) note, "developing new codes as you progress with your analysis provides a more flexible, rich and inclusive … analysis of the information you have collected" (p. 73). During this process, I made use of the constant comparison technique; that is, I continuously compared new data with my existing data, theories, and categories in order to ensure an appropriate fit (Cohen, Manion, & Morrison, 2007), revising and refining my initial coding scheme as needed (Hsieh & Shannon, 2005). My annotations allowed me to comment on a particular section of source material or node. These side notes, in turn, helped me to reflect on my data continuously and record insights, thoughts, questions, ideas, observations, and emergent patterns concurrent with coding (NVivo, n.d.). While I did make use of NVivo's built-in queries to observe word frequencies, I felt more confident reading all comments and exhaustively coding my data. While saturation was often reached several pages into a set of responses, I nevertheless coded each data set, allowing repetitions within nodes. I began with coarsely coding my data into a large number of nodes, and after reflecting on the nodes and annotations made throughout, decided which nodes shared common themes and could be merged together into a higher-level categorical node (Hycner, 1985 as cited in Cohen, Manion, & Morrison, 2007). In other words, by "clustering units of relevant meaning" (Hycner, 1985 as cited in Cohen, Manion, & Morrison, 2007, p. 472) I was able to identify fundamental commonalities, and, through this process, I "eliminated redundancies" (p. 472). For example, the data coded at the nodes "confidence," "confusing, frustrating," "engagement," "enjoyable," "exciting," "interest," "new, fresh," were all clustered under the categorical node "learning experiences" (see Figure 8).



The integration of mathematics and computer programming in math 1ls3 has given me a new outlook on how mathematical functions work and how they can be applied to the real world. They give me a deeper understanding on how systems can be manipulated and how math can relate to the world around us.

The integration of mathematics and computer programming showed me that math goes beyond simple algebraic calculations done by putting pen to paper. I have come to realize that programming helps to solve a multitude of complex programs in a fast, accurate, efficient manner, and can be very useful in real world applications where this efficiency is needed. Ex. Finding the equilibrium of absorption of a drug is helpful for medical purposes and can be done easily with programming, as seen in Lab 4.

My perception of mathematics has changed significantly due to the incorporation of these labs. I able now better able to envision what a career in mathematics may look like. Prior to this, I only had one image of math, number crunching on a calculator and writing down the answers on a piece of paper. However, I am now able to see math as a much more dynamic process with many different applications and career options.

The integration of mathematics and computer programming in 1LS3 has changed the way I view mathematics as I never realized its applicability in the real world

Figure 8: Screenshot of the coding framework in NVivo.

Once all data was coded, I reviewed and reflected on the source content (Cohen, Manion, & Morrison, 2007) in each categorical node, and on all annotations pertaining to the nodes in the category, and then created a memo (NVivo, n.d.) for that category. Each memo summarized the content of the categorical node, included multiple illustrative examples, and connected the results back to my research questions. These memos formed the basis of my Results chapter and were organized into three overarching central themes, which express the "essence" (Cohen, Manion, & Morrison, 2007, p. 472) of each category: modified perceptions of mathematics, enhanced mathematics learning experiences, and unique coding affordances.

3.8 Computational Literacy Framework

To situate my research, I analyzed the results I obtained using diSessa's (2018) literacy framework. diSessa proposed five principles which signal and characterize an emerging literacy. His purpose in developing this framework was motivated by two goals: first, to propose a "big picture" (p. 30) model of computation as a new literacy and second, to provide an analytical framework with which to examine other computational initiatives in education, such as computational thinking and coding.

diSessa's (2018) first principle conceptualizes a literacy as a massive, social and cultural achievement which fundamentally impacts multiple cannons of intellectual enterprise. His remaining four principles are consequences of the development of a new literacy: remediation with a new representational system, reformulation of objects, ideas, and processes, reorganization of the intellectual landscape, and revitalization of the learning atmosphere. diSessa examines his own research in teaching grade 6 students the mathematics of motion, applying the literacy criteria to his data, and discussing the budding of a computational literacy.

Following diSessa's (2018) approach, I adopted his framework for my analysis to investigate the correspondence between my data and diSessa's criteria for an emergent literacy. I examined the categories that emerged from my content analysis and considered the role each one played with respect to diSessa's principles. This enabled me to build a table illustrating the relationship between the results of my study and the anticipated outcomes of a computational literacy. I then considered each of diSessa's literacy principles and chose examples from my data to illustrate each principle and explain how it emerged in our teaching and learning experiences throughout this project.

3.9 Trustworthiness of Study

Lincoln and Guba (1985) outlined four criteria for evaluating the trustworthiness (and therefore worth or value) of naturalistic inquiries: credibility, transferability, dependability, and confirmability. Credibility refers to the confidence with which the conclusions truthfully represent the phenomenon being studied, or reflect the patterns in the data. Transferability determines to what extent the results of a study can be generalized to other comparable situations. Dependability is achieved when the results are consistent and could be reproduced in similar contexts. Confirmability addresses the degree of neutrality of the study.

Lincoln and Guba (1985) describe a set of techniques, which help to achieve their criteria within qualitative research studies. They argue that prolonged engagement and persistent observation help to establish credibility within qualitative research. Throughout this study, I collected and analyzed responses from a large number of students (on average, 900 students consented to have their responses used for research purposes) on four separate occasions over the course of a semester and exhaustively coded each data set. This prolonged engagement with the study and raw data ensured the breadth of my observations and findings, and consequently, of my insights and conclusions. Furthermore, each categorical node contained numerous illustrative examples and/or comments generated independently from numerous students responding to various prompts for feedback. I extensively reviewed, compared, evaluated and re-evaluated students' responses within each node (persistent observation) to guarantee that I had identified all meanings (explicit and implicit) conveyed in each response, and to ensure that I achieved a desired depth in my insights and conclusions. To establish a high degree of transferability, I provided a thick description (Lincoln & Guba, 1985) of my research process—participants, setting, implementation of the intervention, data collection, and content analysis. To establish dependability and confirmability, I maintained a reflexive research journal to provide an audit trail (Lincoln & Guba, 1985) of my research process, my insights, and the development of themes, including illustrative comments from the original source material, all of which helped to lead me to my conclusions.

3.10 Summary

Investigating my research questions as a case study coupled with qualitative content analysis provided valuable insight into how students' conceptual understanding of undergraduate calculus topics changed, and how their mathematical learning experiences were impacted, as a result of engaging with computational modelling activities. Examining my results using diSessa's (2018) analytical framework helped to organize my data and align it with his five well-defined literacy principles. Adopting this frame of reference revealed new insights into my data and enabled a systematic comparison of my research project with related initiatives, such as diSessa's research on teaching sixth grade students the mathematics of motion. Like diSessa, I discovered the unique affordances of remediating calculus with computation, important consequences of reformulating a problem for computation, and the reorganization of the intellectual landscape that occurs when a new literacy is emerging. I experienced a revitalization of my teaching and learning ecology, but also witnessed, and acknowledged the limitations inherent in this new representational system.

Comparing my research with existing efforts, using diSessa's (2018) criteria as a reference, I feel confident that I have come across something valuable. As well, these criteria allow me to situate my study within the larger body of research on computation in mathematics education.

Chapter 4

4 Results

My current research has been guided by the following questions:

1. How does students' conceptual understanding of calculus concepts change in response to working on problem-solving and mathematical modelling activities which incorporate computational thinking?
2. How are students' learning experiences transformed when they explore calculus concepts, ideas and techniques using computational tools and models?

To investigate these questions, I collected students' responses to a series of questions and prompts posed at the end of each of the four computer labs assigned throughout the course. Using a combination of topic and analytic coding to begin my content analysis, I sorted my data into thirteen categories, based on explicit (key words) and implicit (underlying meaning) content. Analyzing the relationships between categories, I was able to further organize my data into three overarching central themes: modified perceptions of mathematics, enhanced mathematics learning experiences, and unique coding affordances. The first two themes address the original research questions and students' reflective comments generated the latter important theme (see Figure 9).

Modified Perceptions of Mathematics

- broader, more representative perspective of the field of mathematics
- enabled meaningful, authentic applications to be incorporated into course activities
- illustrated the relevance and value of mathematical concepts

Enhanced Mathematics Learning Experiences

- interactive learning experiences with immediate feedback provided opportunities to explore, experiment, and play with mathematics
- dynamic visualizations
- transformed affective learning experiences
- tangible feel
- new approach, different perspective
- accommodated various learning styles

Unique Coding Affordances

- elevated problem solving capabilities beyond traditional limits
- problem solving became more efficient, less tedious
- offered unique advantages over ready-made applications
- physical coding mechanics provided numerous benefits

Figure 9: Results of qualitative content analysis.

4.1 Modified Perceptions of Mathematics

A computational thinking approach challenges traditional views of what mathematics is and what mathematicians do. In particular, these views are often quite entrenched with regards to the content that is taught in calculus (both in high school and university), as well as in the teaching methods and approaches.

Due to certain affordances of algebraic representations in teaching practice, algebraic techniques are often emphasized and viewed as the most sophisticated approach to problem solving. However, in most cases, realistic data from outside of a theoretical mathematics course cannot be analyzed with concepts and tools developed for continuous functions. Instead, these theoretical concepts and tools are reformulated in discrete terms, so that numerical approaches can be used to analyze the data and build appropriate models.

Integrating computer programming into problem-solving activities in calculus allows students to further develop their knowledge, skill, and appreciation of different mathematical approaches. The traditional "rule of three" (algebraic, geometric and numeric approaches to learning and understanding mathematical content), sometimes augmented to the "rule of four" (by adding a verbal approach), are enriched by adding an important, far-reaching, computational approach. Together, these approaches reformulate calculus concepts developed for continuous functions into discrete analogues, which not only reinforce students' understanding, but also more readily allow calculus tools to be incorporated into problem solving in other disciplines.

4.1.1 Broader, More Representative Perspective of the Field of Mathematics

Students reported that integrating coding with calculus concepts broadened their perspective of mathematics from a discipline that leaves no room for interpretation or inquiry, to one that invites investigations, explorations, new techniques and approaches, as well as one which supports inquiry-based thinking. They stated that the coding activities demonstrated an interdisciplinary approach to mathematics and allowed mathematical concepts and tools to be effectively integrated with other disciplines to help solve complex problems.

Sample of students' comments:

> *I liked how open the lab was and how experimentation was openly encouraged. It helped me think about the material in a deeper way.*

> *My perception of mathematics has changed significantly due to the incorporation of these labs. I able now better able to envision what a career in mathematics may look like. Prior to this, I only had one image of math, number crunching on a calculator and writing down the answers on a piece of paper. However, I am now able to see math as a much more dynamic process with many different applications and career options.*

An advantage to this is it expands new ideas and ways to tackle a certain question. Also it helps broaden our understanding of how mathematics works beyond the scope of traditional mathematics alone.

Through the use of computer science and coding, the module depicted the importance of mathematics in real world scenarios and how technology and mathematics can be incorporated hand in hand to accomplish complex goals.

4.1.2 Enabled Meaningful, Authentic Applications to be Incorporated into Course Activities

Students reported that integrating coding with mathematics afforded unique opportunities to explore and analyze authentic applications, which may be inaccessible or impractical to consider otherwise. For example, in Math 1LS3, students explored the SEIR (susceptible–exposed–infected–recovered) model to understand the dynamics of the recent EBOLA virus outbreak in several African countries. This complex model would usually be studied in a second or third year differential equations course; however, by remediating the system of differential equations computationally, students were able to extend Euler’s method from a single first-order differential equation to a system of four first-order differential equations. Furthermore, they were able to run thousands of iterations in a fraction of a second to explore long-term behaviour of the spread of the virus. Students were then able to modify parameters in the SEIR model to determine their individual effects on the outbreak, that is, which parameters have to change—and to what values—in order for the outbreak to be controlled. In previous semesters of Math 1LS3, the classic predator-prey model was only explored qualitatively, since algebraic solutions are known to be impossible to obtain (except in some very special cases). However, again by intuitive extension, students are able to apply Euler’s method to a system of equations to uncover periodic behaviour of solutions. It is highly unlikely that applying Euler’s method by hand, using several iterations, would uncover this pattern.

Students described how their interest and engagement increased when they saw that an abstract mathematical concept could be applied to effectively investigate an authentic, real-world problem. This helped provide tangibility to the theoretical mathematical concepts, or as one student stated, "it grounds the math." Moreover, by incorporating meaningful applications, students' analytical activities more closely aligned with practices in the field, which students said increased the value of the material they were learning and allowed them to have a realistic perspective of mathematical research methods and what a career in mathematics might look like.

Sample of students' reflections:

> *... I also like how the problems are based on real-world scenarios, as opposed to dry mathematical questions. It makes you look at math in a new way, and it gives it significance/importance.*
>
> *I feel that the applications in this Lab help me to make bridges between concepts and usefulness in real life. I feel that once I establish that connection, I understand concepts much better and am able to answer problems easier.*
>
> *... we can assess the more convoluted functions of real-life, rather than sticking to simplistic models. In-class instruction becomes so much more relevant to real life.*
>
> *...coding was interesting and allowed for me to see concepts used in more realistic scenarios, coding allowed for massive/incalculable scenarios of eulers method etc. to be done (real life trends that are too large/complex to be done by hand). This gave me a greater appreciation for the math concepts and how they apply to reality.*

4.1.3 Illustrated the Relevance and Value of Mathematical Concepts

Students found that the coding exercises effectively illustrated the relevance of the mathematical skills and concepts they were studying. In particular, they noted that integrating coding with numerical approaches, such as approximating a definite integral using a Riemann sum or estimating a solution to an initial value problem using Euler's method, effectively demonstrated the value of these concepts and techniques, which were formerly perceived as inferior to algebraic methods.

For example, students often perceive using the definition of the derivative to find the instantaneous rate of change of a function as a naïve method, only used before learning specific differentiation rules. A root cause of this view can be traced to the way derivatives are covered in high school. For instance, Kajander and Lovric (2009) show that, by not using the definition of the derivative to interpret and visualize the tangent line to the graph of a function, grade 12 students develop a number of misconceptions (such as "a tangent touches the graph at one point, but cannot cross it" (p. 175)), which hinder their understanding and progress.

In Math 1LS3, we presented students with a function, represented numerically in Excel, which recorded the number of individuals hospitalized with influenza in Canada each month over the course of several years. Students were asked to determine the rate of change of serious cases of influenza in Canada from this function, which was represented as a large set of discrete values. Through this example (and several other examples involving functions represented numerically), students realized that in many applications, rates of change are approximated using *average* rates of change—that is, difference quotients—since differentiation rules apply only to continuous functions (and not even to all continuous functions!). Furthermore, by analyzing rates of change using code, these calculations can be done quickly and efficiently, simultaneously producing a visualization of the data set and the approximate rates of change.

Sample of students' comments:

> *Using mathematic techniques via coding that would be difficult to do in detail by traditional methods (Ex. Euler's method, Riemann Sums) made me appreciate the techniques much more than if I had not seen the capabilities of the techniques, and only saw a couple of iterations of the techniques.*
>
> *I feel that coding gives us a sense of real-life mathematics and its usefulness that traditional mathematics would not. Using mathematical modelling of functions through codes in real life makes us feel like we can use this skill in our workplaces in the future too and it gives the learned content new value.*

> *…it was effective in making math into something more tangible; something that can be extended beyond the classroom. It felt nice knowing what actual researchers in the life science do with their data and how they manipulate it.*

> *I believe that seeing mathematics and computer programming joined together in such a manner has allowed me to appreciate the value of mathematics as a tool for modelling data that can then efficiently be processed using computer programming.*

4.2 Enhanced Mathematics Learning Experiences

Adopting a computational thinking approach enabled students to incorporate powerful computational tools into their mathematical problem solving. The categories in this section reflect particular, and often unique, affordances of computer coding environments.

4.2.1 Interactive Learning Experiences with Immediate Feedback Provided Opportunities to Explore, Experiment, and Play with Mathematics

Students reported that the coding environment offered a dynamic and interactive learning experience during which they could effectively explore and analyze mathematical models and techniques in innovative ways. Students also noted feeling more interested and engaged in the activities, and stated that they felt they achieved a more comprehensive understanding of a concept when they were given the opportunity to actively interact with the components of a problem.

Moreover, running computer code provided immediate feedback, which students said required of them to critically evaluate their work to recognize the nature of any errors (mathematical or coding), and allowed them to effectively make and test their corrections. Students stated that receiving this constant formative feedback helped shape their mathematical understanding and improved their confidence.

Students reported that coding activities gave them the freedom to independently explore the mathematical content in ways that were meaningful and relevant for them. They appreciated the ability to ask hypothetical, "what if" questions, test their predictions, and receive immediate feedback, which helped them develop deeper insights and intuition, as well as clarify any confusions. Students noted that the opportunity to experiment with different scenarios allowed them to gain a more comprehensive understanding of the mathematical relationships and concepts.

Sample of students' reflections:

> *Traditional instruction lacks the immersive and interactive nature that coding gives you. When I'm coding, I personally feel very engaged with the source material. It's as if this is my project, it's a problem and a journey, as I endeavour to solve it.*
>
> *This module allowed me to go back and fix my mistakes, if any. This allowed for me to think critically and be able to rectify my mistake, and that embedded the fundamentals of the mathematics within me.*
>
> *In traditional mathematics you do not have the opportunity, nor the time to explore different ways to come up with the same mathematical answer. Coding allows us to make mistakes and understand where those mistakes came from, and gives us an opportunity to THINK.*
>
> *Coding lets me see how changing different things about a problem affects it and it allows me to work by trial and error based on what I personally need to do to understand. While traditional instruction is still much more organized and delivers information more directly, coding is a great way to apply new knowledge and clarify confusion through experimentation.*

4.2.2 Dynamic Visualizations

One of the most frequently reported comments was that students appreciated the dynamic visualizations that combining code with their mathematical explorations could provide. These visualizations included graphical comparisons between numerical approximations and theoretical solutions, extensive lists of numbers from which a pattern is to be discerned, and areas of regions bounded by curves. Students reported that interacting with these visual representations enhanced their conceptual understanding of the corresponding mathematical ideas. While students recognized other programs and applications (such as Desmos) could provide visualizations as well, by creating their own code in Jupyter Notebook, students could perform multiple analyses at the same time; that is, they could create an algorithm to produce the desired quantitative output, and also write a code for a visual representation to be generated simultaneously. This code could then easily be copied, pasted, and modified in a new cell to perform a similar analysis under different conditions. Students noted that prepackaged programs are more limited in their capabilities, whereas coding offered them full control over what their particular program does.

Sample of students' comments:

> *The activities in Lab 3 that I found most effective at enhancing your understanding of the mathematical concepts were the ability to create visual depictions of the math. This helped me fully see consequences of certain actions and helped me fully comprehend the effects, thus increasing my overall understanding.*
>
> *Without a visual representation and physically playing around with the numbers, I could never have completely understood the concept [of Euler's method].*
>
> *When coding, you can have a visual representation of your work making it easier to identify mistakes.*

4.2.3 Transformed Affective Learning Experiences

Students reported that integrating coding activities and calculus concepts enhanced their learning experiences in a variety of ways. They stated that the approach felt "new," "fresh," and "modern," which increased their interest during the problem-solving activities. They also remarked that incorporating authentic applications made mathematics more stimulating and relevant.

Students noted that the activities invited innovative approaches to problem solving, which made the mathematical content more exciting and enjoyable to study. Most students did not have any high school experience with coding, so having this new tool to help them answer mathematical questions was indeed novel to them. They found that the dynamic visualizations, as well as the automation of tedious routine calculations, increased their overall enjoyment of the activities.

Students reported feeling more engaged in mathematical problem solving since the coding activities required their full attention and active interaction. As well, they noted that the immediate feedback they received was rewarding and motivated them to explore concepts further.

Many students felt that using code to explore mathematical concepts opened up a creative space for problem solving that was not previously available (and unfortunately not available in high school). They appreciated that there were multiple ways they could approach a problem, and remarked that coding offered a level of flexibility not typically offered within other mathematical problem-solving environments.

Students reported that the mathematical coding activities encouraged meaningful, productive peer collaborations. They noted that engaging in creative struggles, discussing the material, and comparing different coding approaches with their peers helped facilitate understanding as well as broaden their social network.

Sample of students' reflections:

> *Overall I really enjoyed the coding exercises in this module, it definitely increased my motivation and confidence, but more importantly my understanding of the content learned in class.*
>
> *I thoroughly enjoyed the computer labs as they allowed me to interact with mathematics in a new way.*
>
> *I found that this coding lab made me more engaged in the math content. It made the work being done seem less mechanical and gave more purpose to the work.*

> *I personally think the advantages of these coding labs come in the form of flexibility, as it allows students to be more creative in the ways that they get to the final answer.*

> *I believe that these coding activities were a great concept to add into the course and it was a great way for many students to collaborate and work together. Working with other individuals helped me understand concepts better as we explained different things to each other.*

4.2.4 Tangible Feel

Students reported that interacting with mathematical models and algorithms using code aided them in forming concrete representations of the abstract mathematical concepts. They found that coding helped ground the mathematics for them, and that the abstract concepts became more real and tangible through the coding activities. For example, many students reported that the definition of semilog and double-log plots did not make much sense to them until they had the opportunity to interact with the code for these plots, and with the plots themselves. These interactions helped clarify the definition by "seeing," that is, by providing a tangible feel to this concept of using logarithmic scales on coordinate axes (instead of the usual linear scales) that would otherwise remain abstract and have a theoretical feel only.

Sample of students' comments:

> *I liked the visual explanation of the Intermediate Value Theorem, and I liked being able to manipulate code in order to learn about it, and truly understand what it meant. By working through a code example of a theorem, you make it less of an abstract idea, and more of a practical application.*

> *With the integration of computer programming, I feel like these concepts and problems become much more tangible. I think it is likely due to the fact that the labs provide a 'hands on' aspect you wouldn't normally get.*

> *The integration of math and computer programming has made math more real to me. It made the concepts much less abstract to me and more tangible.*

> *Mathematics has always been an abstract concept to comprehend, however, the addition of computer programming allows the concepts to be grounded in practical applications that can be understood and manipulated.*

4.2.5 New Approach, Different Perspective

Students reported that integrating coding with mathematics offered a new perspective on mathematical models and concepts, and invited multiple approaches to problem solving. They found it interesting to compare how they would solve a problem algebraically to how they would reformulate it as an algorithm so that they can use computer code. For example, when required to determine the area between two curves, students found it interesting, and eye-opening, to contrast how they would approach the problem algebraically using the Fundamental Theorem of Calculus to how they would approach it numerically, by creating a program in Python to estimate the area using Riemann sums. In another example, students found that using an alternative method to find and analyze critical numbers (in their code, students used the Intermediate Value Theorem), helped to solidify the definition of a critical number, the algorithm for finding critical numbers, and the First Derivative Test.

Students realized that when they explored a concept using multiple representations, they were able to benefit from various affordances, and fill in the gaps stemming from the limitations of a single representation, by considering complementary representations.

Sample of students' reflections:

> *Not only did the questions reinforce my existing knowledge, but it also prompted me to assess the questions in a different manner and encouraged critical thinking.*

> *…builds mathematical understanding in a unique way, creating new pathways for the brain to solve mathematical problems.*

Overall the process of working on behind the scenes for these functions helped me look at them from a different angle and extend my understanding from the lecture.

I find when I'm doing the actual questions myself I think of how the code was configured, and it helps me understanding what I am doing.

4.2.6 Accommodated Various Learning Styles

Students reported that analyzing mathematical models and algorithms using code provided differentiated learning opportunities, which supported a variety of learning styles. For example, many students reported being visual learners and appreciated how running code produced rich visualizations, which supported their learning in a way that non-coding activities (or their textbook) could not. They further noted that the labs enabled them to adopt an interactive, hands-on approach to their learning, which was especially beneficial to tactile, kinetic, and visual learners.

Students also remarked that exploring mathematical concepts using code encouraged independent learning more than traditional problem-solving activities. For instance, they reported that they were less afraid of making mistakes, as they knew that Python would spot the mistakes right away, and force them to fix their code, or to modify their mathematical approach, without a lot of extra work on their part. Students also appreciated that coding allowed them to personalize their learning experiences by independently exploring concepts at their own pace in ways that were meaningful to them.

Sample of students' comments:

Some people learn things differently than others and many of them, like myself, learn by doing things. We need to see the mathematical concepts applied in front of us and need some hands on experience with those concepts. By integrating coding and math, people like me can manipulate equations in whichever way we like and see the real time consequences of our actions.

The computer lab/module did a great job in teaching the course material from different angles. Different students learn in different ways and it is very difficult to teach a concept that will be understood by all of these students. This module was effective in showing students other ways of learning that may have not been clear prior.

Coding lets me see how changing different things about a problem affects it and it allows me to work by trial and error based on what I personally need to do to understand.

Using the coding software also allows students to see different representations of math (graphs, tables, equations), and choose which one they understand best.

4.3 Unique Coding Affordances

Analyzing the connections between categories revealed three overarching themes. The first two themes addressed the original research questions, whereas certain categories suggested a third theme: unique affordances of exploring mathematics with computer code.

4.3.1 Elevated Problem-Solving Capabilities Beyond Traditional Limits

Students recognized that while, theoretically, they could do the computations they were coding by hand, the complexity of the models they were working with, as well as the sheer number of calculations or iterations of the method required to obtain a meaningful result, would make these calculations impractical or impossible to obtain in a reasonable time frame without integrating computer coding to some degree.

By adopting a blended approach of using theoretical, algebraic, and computational techniques, multiple constraints were removed and students reported experiencing more freedom to explore even the most complex situations. They said that they were motivated to ask deeper theoretical questions and further explore the problems and mathematical concepts, without the burden of technical computations (which indeed seems to be a burden for many students!) restricting their time and mental energy.

For example, using Euler’s method to estimate the solution to an initial value problem often requires a very large number of iterations to achieve a meaningful result. Students complain that these calculations are repetitive, tedious and error-prone (and they are!), even beyond two or three steps. Furthermore, the simplicity and versatility of this estimation method (which was even featured in the movie *Hidden Figures* (Melfi, 2016)) is obscured by the cumbersome calculations underlying it. Remediating Euler’s method with code removes the tediousness of the technical calculations and allows students to apply it to a system of any number of first-order differential equations, where initial values are given.

Sample of students’ reflections:

> *I think that the integration of coding in mathematics helps add extensions to what is possible from instruction alone. It allows you to explore and "play" with concepts in a way that couldn’t be possible without the use of technology.*
>
> *By allowing mathematical calculations to occur that would not be possible by algebra there is a new avenue of possibilities made available in what can be calculated.*
>
> *Since the computer is doing all of the calculating for you, you aren’t limited by the amount of time it would take to solve something. Due to this, you can incorporate real data and use concepts from class to work with the data and see the importance of different math concepts in everyday life.*

4.3.2 Problem Solving Became More Efficient, Less Tedious

Most students—even those who reported not enjoying the computer labs—appreciated how efficiently (and correctly) complex calculations could be done almost instantaneously in Python 3. They remarked that they could focus more on the conceptual understanding and ask deeper theoretical questions when they knew that they would not be facing tedious, routine procedural calculations. As well, students felt encouraged to fully explore mathematical concepts, such as extending Euler’s method to generate approximate solutions to a system of differential equations, rather than just to a single equation, without noticeably increasing computation time. Students reported that this efficiency in computation increased their interest and engagement in mathematical problem solving.

Sample of students' comments:

> *It saves time on little calculations so that students can see the bigger picture without getting caught up on minor details.*
>
> *Coding makes mathematical ideas far more interesting as it provides a more efficient way to explore the possibilities of a function as well explore other mathematical ideas."*
>
> *…coding provided me more time to further explore the nuances within the questions themselves.*

4.3.3 Offered Unique Advantages Over Ready-Made Applications

Students identified several affordances of integrating coding and mathematics, which extend beyond what non-coding technology can offer. For example, they stated that coding offers more control over their explorations and provides a greater feeling of satisfaction and accomplishment when they obtained the desired result. Students also noted using a coding language possesses higher capabilities and greater versatility than using a prepackaged application, thus eliminating the need for several different technologies to explore a problem or concept since multiple analyses can be performed simultaneously within a coding environment.

Furthermore, students remarked that working with code arranged in cells helped to organize and store their work so they had a record of their previous results and could run new simulations or perform further analyses without starting over from scratch. For example, one activity invited students to explore the solution to a modified logistic differential equation describing the population dynamics of a caribou population in Northern Alberta, starting from a given initial population size. Once the initial coding template for applying Euler's method was created, students copied, pasted, and modified the code to explore various scenarios, revealing (experimentally) the existential threshold and carrying capacity for this population, as well as the stability of these equilibrium solutions—topics which are algebraically explored in a second year differential equations course.

Sample of students' reflections:

> *… students have greater agency and can create pretty much anything whereas in traditional mathematics instruction, there are much more limits and its more structured.*
>
> *Manipulating the code to run equations and seeing an actual result was a very rewarding experience and I really felt more confidence with the problem I was solving.*

4.3.4 Physical Coding Mechanics Provided Numerous Benefits

Students identified several aspects of the physical coding process, which enhanced their learning and understanding. Since coding languages are very particular in terms of their syntax, students reported that they needed to think critically throughout each step of the problem-solving process, paying close attention to detail, in order to produce a fully functioning program. Students commented that this heightened focus and deeper thinking helped them understand the relationships between components of the problem and their code, and enriched their understanding of the logic underlying the mathematical processes involved.

Additionally, students reported that the process of deconstructing a problem and reformulating it for computation required a thorough understanding of the mathematical concepts, relationships, and algorithms underpinning the exercises. They remarked that the process of deconstructing the mathematical ideas (e.g., models, techniques) into basic elements helped reduce the complexity of the problem and promoted a thorough understanding of the relationships between components.

Reformulating a problem for computation involved translating textbook models and algorithms into code versions, which students stated helped them form a stronger connection to the conceptual ideas. For example, one student reported being confused when calculating the next value of the state variable using Euler’s method, consistently forgetting which values they should be using in the formula. The process of converting the algorithm in the textbook to Python code helped to clarify the reasoning behind the recursive pattern and improved the student’s overall understanding of Euler’s method. Moreover, students mentioned that the active process of simply typing code helped them to internalize definitions and concepts.

Sample of students’ comments:

> *Because the code requires you to define everything and practically explain all the variables and how they relate to each other, it makes you think critically even when solving the smallest math problems.*
>
> *Just doing the programming helped me to internalize the math being done and helped me understand it better.*
>
> *… coding out individual steps of the Euler's method demonstrated the specific mechanisms behind the method and lead me to further understanding.*
>
> *I found that in my own head I was able to break down the intermediate value theorem in a different way, piece by piece and as such my understanding of the concept as a whole (and it’s applications) were improved.*
>
> *Coding allows students to think critically in terms of communication - how to explain a mathematical process in objective terms. This is how the code input tells the computer program what to do. By going through this process, students understand the math processes more deeply as they are now able to describe it in objective, systematic ways that even a computer would understand.*

4.4 Summary

Conducting a systematic qualitative content analysis helped to organize the raw data into categories and reveal three central themes: modified perceptions of mathematics, enhanced mathematics learning experiences, and unique coding affordances.

In the next chapter, I use diSessa's (2018) literacy principles as a theoretical lens through which I examine my results more in depth. diSessa developed these principles in part to characterize and identify an emergent computational literacy, but also as an analytical framework with which to analyze contemporary movements of computation in education, such as computational thinking and coding. diSessa used this criteria to analyze his work with teaching grade 6 children the mathematics of motion. I adopt a similar strategy to analyze my approach of integrating computational thinking, coding, and mathematical problem solving into an applied undergraduate calculus course.

Chapter 5

5 Analysis

In this chapter, I further analyze the data that I obtained (and summarized in the Results chapter) to help situate my research within the broader context of a computational literacy. As a framework for my analysis, I used diSessa's (2018) five literacy principles, which he developed to signal and characterize a new (in this case, computational) literacy (see Figure 10). This lens also serves as a frame of reference which diSessa uses to analyze computational initiatives in education, such as computational thinking and coding.

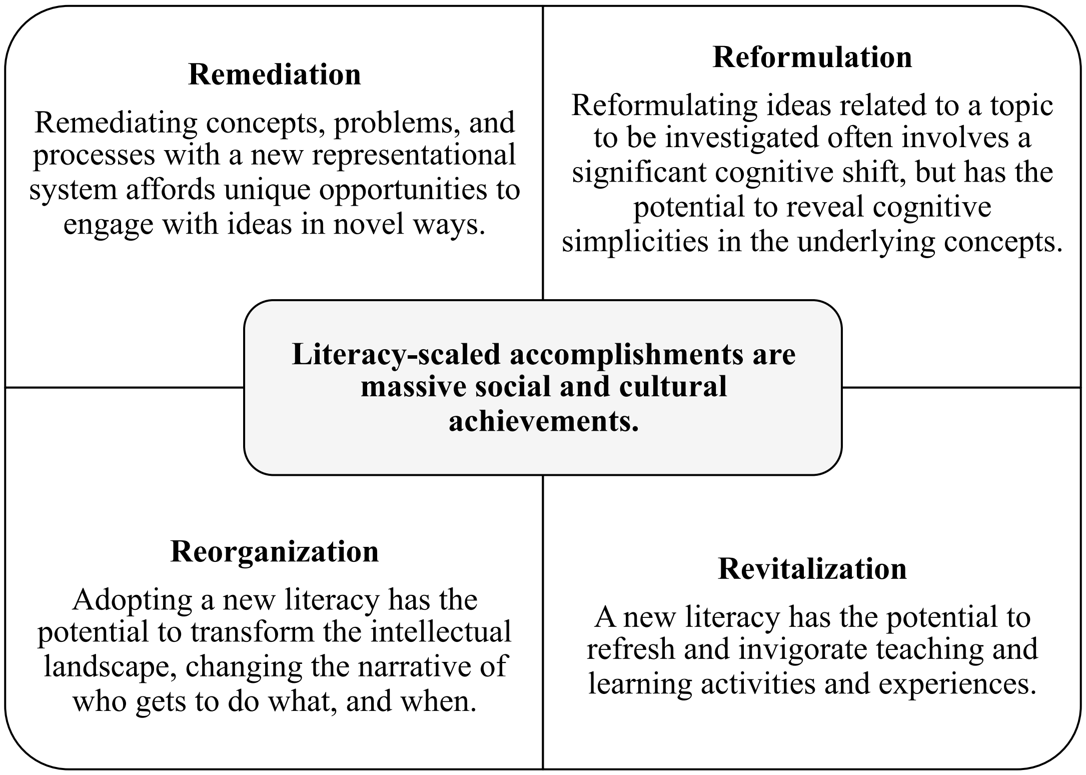

Remediation
Remediating concepts, problems, and processes with a new representational system affords unique opportunities to engage with ideas in novel ways.
Reformulation
Reformulating ideas related to a topic to be investigated often involves a significant cognitive shift, but has the potential to reveal cognitive simplicities in the underlying concepts.
Literacy-scaled accomplishments are massive social and cultural achievements.
Reorganization
Adopting a new literacy has the potential to transform the intellectual landscape, changing the narrative of who gets to do what, and when.
Revitalization
A new literacy has the potential to refresh and invigorate teaching and learning activities and experiences.

Figure 10: diSessa's (2018) literacy framework.

The chart in Figure 11 illustrates the mapping between my results (i.e., the thirteen categories I identified in the Results chapter) and diSessa's (2018) four principles of a new literacy: remediation, reformulation, reorganization, and revitalization. In the discussion that follows, I do not revisit nor examine each category in detail; instead, I delineate these four principles and illustrate each with salient examples from my data. I conclude my analysis by considering how my results provide evidence of a new literacy, as defined by diSessa.

	Remediation	Reformulation	Reorganization	Revitalization
more representative perspective of the field of mathematics				✓
enabled meaningful, authentic applications to be incorporated into course activities	✓		✓	✓
illustrated the relevance and value of mathematical concepts	✓			✓
interactive learning experiences provided opportunities to explore, experiment, play with mathematics	✓			✓
dynamic visualizations	✓			
transformed affective learning experiences				✓
provided a tangible feel to abstract concepts	✓	✓		
new approach, different perspective		✓	✓	
accommodated various learning styles			✓	✓
elevated problem-solving capabilities beyond traditional limits	✓		✓	
problem solving became more efficient, less tedious	✓		✓	
offered unique advantages over ready-made	✓			

applications				
numerous benefits realized through physical coding process	✓	✓		

Figure 11: Mapping categories from Results chapter to diSessa's (2018) principles.

5.1 Remediation

Concurrent with the acquisition of a new literacy is the development and adoption of an appropriate representational system used to remediate ideas, processes, and problems and describe, analyze, and explore them in terms of the new literacy (diSessa, 2018). The mass appropriation of a new representational system will demonstrate "distinctive and critical strengths, but also limitations and blind spots, and, thus, a possible complementarity with other forms of representation" (diSessa, 2018, p. 7). diSessa (2018) emphasizes the affordances realized by remediating concepts within a new representational system, and explains in which ways remediation contributes to a transformative shift in how we think about ideas, engage with concepts, develop our conceptual understanding, and solve problems.

In Math 1LS3, we remediated our calculus concepts with a computational representational system, which allowed us to explore problems with computer code. Here, I focus on several unique affordances of using computer code (and thus, computational thinking) to explore calculus concepts, as experienced and reported by students in Math 1LS3.

5.1.1 Advantages of a Computational Representational System

One of the most noteworthy observations frequently reported by students was that remediating calculus concepts with computer code enabled them to effectively incorporate computer technology into their investigations, which offered significant technical advantages. They remarked that exploring models, concepts, and algorithms with computer code helped optimize their problem-solving activities by enabling numerous, technically complex, calculations to be carried out almost instantaneously, and producing consistent, accurate results. Automating the numerical calculations eliminated (or significantly reduced) technical difficulties, and enabled students to expand their explorations beyond the constraints imposed by using non-computational tools, and thus motivated them to thoroughly investigate mathematical concepts (for example, by running multiple simulations simply by changing a few parameters to explore hypothetical alternative cases).

Students reported that the ability to efficiently explore their "what if" questions and receive immediate feedback helped them develop a more comprehensive understanding of the relationships between quantities, the behaviour of models, and the logical structure of the mathematical techniques. Furthermore, they noticed that they were able to focus more on the bigger picture and developing their conceptual understanding of important mathematical ideas when their mental energy was not expended on lengthy, complex, repetitive calculations.

5.1.2 Example: Euler's Method

Euler's method is a numerical approach used to approximate a solution to a first-order differential equation, when an initial condition is given (this is known as an "initial value problem").

For the initial value problem, consisting of a differential equation and an initial condition,

$$\frac{dy}{dx} = G(x, y), \qquad y(x_0) = y_0$$

Euler's algorithm is given by two recurrence relations, one for the independent variable, and the other for the unknown function:

$$\begin{aligned} x_{i+1} &= x_i + h \\ y_{i+1} &= y_i + G(x_i, y_i) \cdot h \end{aligned}$$

where $i = 1, 2, 3, 4, \ldots$ and h is the step size.

In a typical calculus course, students are asked to approximate the solution to a simple first-order initial value problem by applying Euler's method for a maximum of three to four steps, thus obtaining a superficial, uninteresting, yet easy to compute by hand, result (see below).

Problem:

Given the initial value problem, $dP/dt = 0.02P(1 - P/2000)$, where $P(0) = 120$, estimate the value of $P(1)$ using Euler's method and a step-size of 0.5. The time t is given in months.

Solution:

In this case, $h = 0.5$, $t_0 = 0$ and $P_0 = 120$, and the Euler's method algorithm is given by

$$t_{i+1} = t_i + 0.5$$
$$P_{i+1} = P_i + 0.02P_i(1 - P_i/2000) \cdot 0.5$$

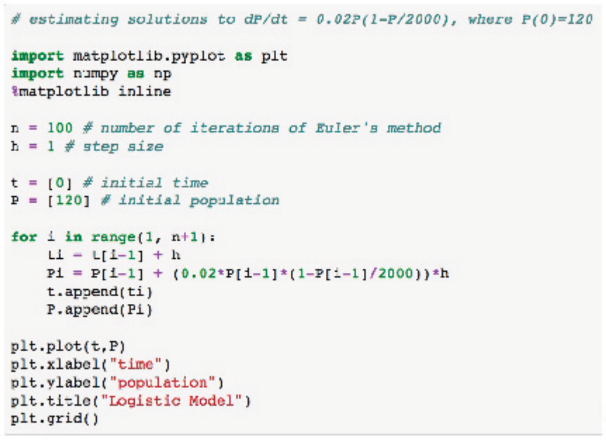
The actual calculations proceed as follows:

$$t_1 = 0 + 0.5 = 0.5$$
$$P_1 = 120 + 0.02(120)(1 - 120/2000)(0.5) \approx 122$$

$$t_2 = 0.5 + 0.5 = 1$$
$$P_2 = 122 + 0.02(122)(1 - 122/2000)(0.5) \approx 125$$

The value of $P(1)$ is approximately 125.

By exploring this initial value problem with code, students are able to investigate the behaviour of the model over a longer period (say, over many months or even years) as well as increase the accuracy of their estimations (by decreasing the step size), an activity that would be unwieldy without computer technology.



```
# estimating solutions to dP/dt = 0.02P(1-P/2000), where P(0)=120

import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

n = 100 # number of iterations of Euler's method
h = 1 # step size

t = [0] # initial time
P = [120] # initial population

for i in range(1, n+1):
    ti = t[i-1] + h
    Pi = P[i-1] + (0.02*P[i-1]*(1-P[i-1]/2000))*h
    t.append(ti)
    P.append(Pi)

plt.plot(t,P)
plt.xlabel("time")
plt.ylabel("population")
plt.title("Logistic Model")
plt.grid()
```

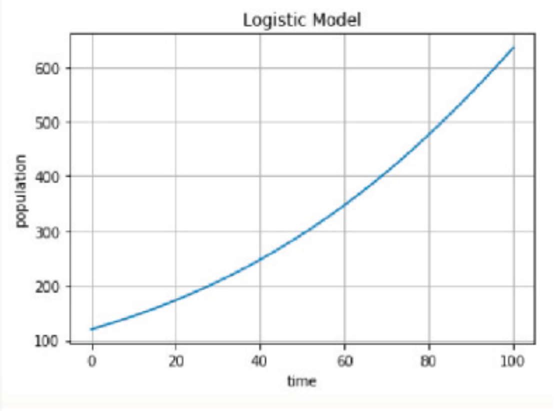
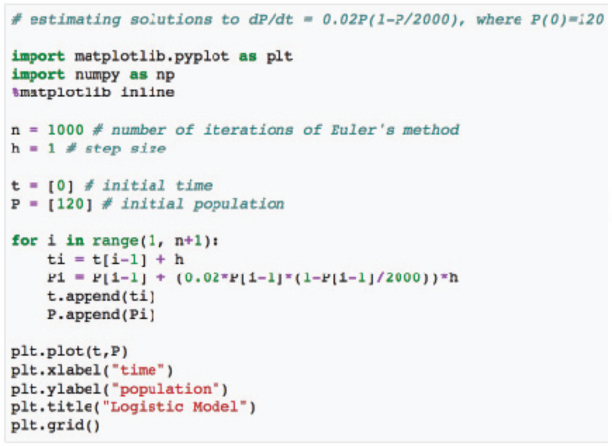


Figure 12: Identifying a pattern in the solution obtained using Euler's method. Note: An insufficient number of steps suggests a realistically unsustainable exponential growth.

There is another, even more important aspect—by seeing only a few steps of an iteration for a function, it might be hard to identify a pattern, or the pattern that is suggested might be misleading, giving an inaccurate solution. For instance, the first few steps of Euler's method might suggest exponential growth (Figure 12), which is not sustainable in the long run. Instead, the initial exponential growth is often followed by a slowdown, resulting in a logistic, limited growth pattern, which becomes visible only if Euler's method is run for a large(r) number of steps (Figure 13).



```
# estimating solutions to dP/dt = 0.02P(1-P/2000), where P(0)=120

import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

n = 1000 # number of iterations of Euler's method
h = 1 # step size

t = [0] # initial time
P = [120] # initial population

for i in range(1, n+1):
    ti = t[i-1] + h
    Pi = P[i-1] + (0.02*P[i-1]*(1-P[i-1]/2000))*h
    t.append(ti)
    P.append(Pi)

plt.plot(t,P)
plt.xlabel("time")
plt.ylabel("population")
plt.title("Logistic Model")
plt.grid()
```

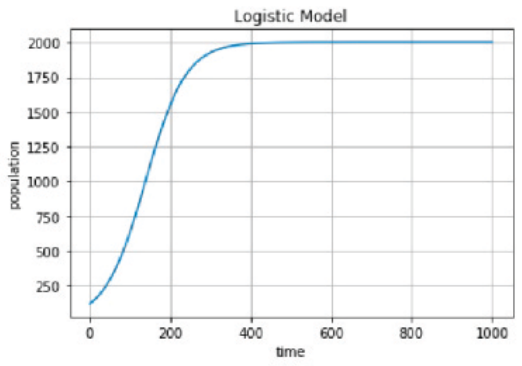


Figure 13: Identifying a pattern in the solution obtained using Euler's method. Note: Logistical (limited growth) pattern is revealed only after the method is run with a large number of steps (and thus over a longer period of time).

5.2 Reformulation

When engaging with a new literacy, all concepts, problems, and processes related to an investigation must be reformulated appropriately so they may be effectively remediated with the new representational system. In Math 1LS3, reformulating a calculus problem expressed algebraically so that it can be represented, analyzed, and solved computationally requires two main processes of computational thinking: abstraction and automation. Reformulating a problem as an algorithm (so that it can be coded) involves deconstructing the problem into basic components (elements), analyzing the relationships between components, and then designing an appropriate computational model in order to automate a solution. This reformulation process requires an in-depth conceptual understanding of all aspects of a problem, and a strong enough familiarity with both formulations that one can effectively translate between two representational systems. As diSessa (2018) explains, reformulating problems often requires a significant cognitive shift (as I discuss below), however this process also has the potential to reveal "surprising cognitive simplicities and when they align with a powerful representational change… learning becomes amazingly transformed, faster, and easier" (p. 15).

In Math 1LS3, remediating a problem with a computational representational system required us to reformulate the problem numerically, that is, we considered discrete manifestations of all concepts and calculations involved. This process is straightforward for those mathematical problems where a numerical problem-solving approach has already been established (for example, Riemann sums, Euler's method, or discrete-time dynamical systems). This numerical representation (model) was then reformulated again so that it could be analyzed using a computational representational system. The following example illustrates the two-step reformulation process we used to remediate our mathematical problems with computation.

5.2.1 Example: Riemann Sums

In covering integral calculus in university courses, a significant amount of time is spent on techniques of integration, that is, on algebraic methods of evaluating definite and indefinite integrals.

A definite integral is defined as the limit of a Riemann sum:

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x, \quad \Delta x = \frac{b-a}{n}$$

where x_i^* is any sample point in the subinterval $[x_{i-1}, x_i]$.

A definite integral can be interpreted as the net or signed area of the region bounded by the graph of a function and the horizontal axis over a finite interval (see Figure 14).

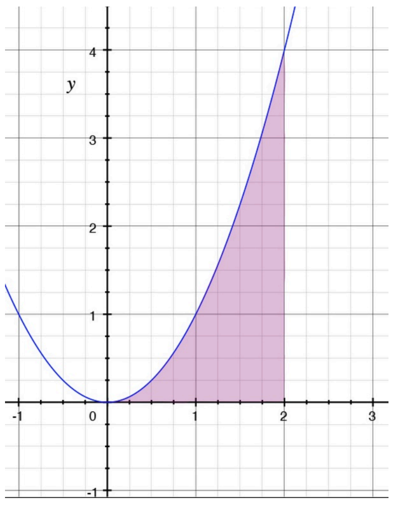
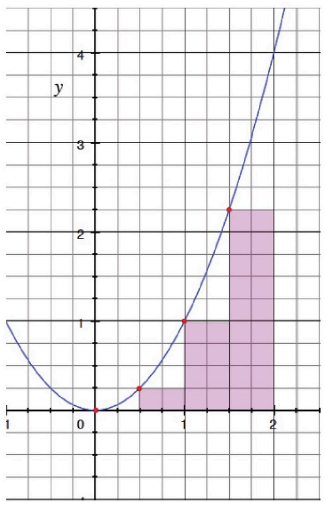


Figure 14: Shaded is the region bounded by the graph of $f(x) = x^2$ and the horizontal axis, defined over the finite interval $[0, 2]$. The area of this region is determined by evaluating the definite integral $\int_0^2 x^2\, dx$.

The area of this irregular region (irregular in the sense that we do not have a ready-made formula established for its area) can be approximated using rectangles, whose areas are easy to compute ("area of a rectangle equals length times width"). The sum of the areas of these rectangles, that is, a Riemann sum, estimates the area of the bounded region, and at the same time, the value of the definite integral (see Figure 15).

To use this approach, we first decide on how many rectangles we will use and then compute the fixed width of each rectangle. After that, we need to decide how to select the heights of the rectangles. Two common choices involve using values at the left-endpoints or right-endpoints of each subinterval.



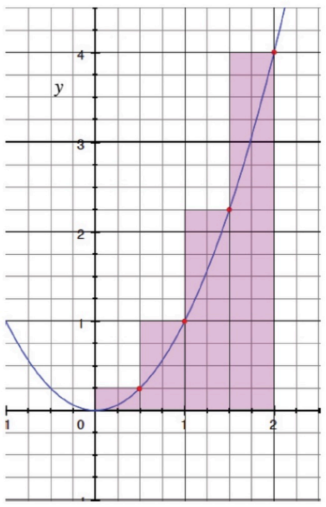


Figure 15: The sums of the areas of approximating rectangles are used to approximate $\int_0^2 x^2 dx$. The figure on the left illustrates the approximating rectangles obtained using left-endpoints; the figure on the right illustrates the approximating rectangles obtained using right-endpoints.

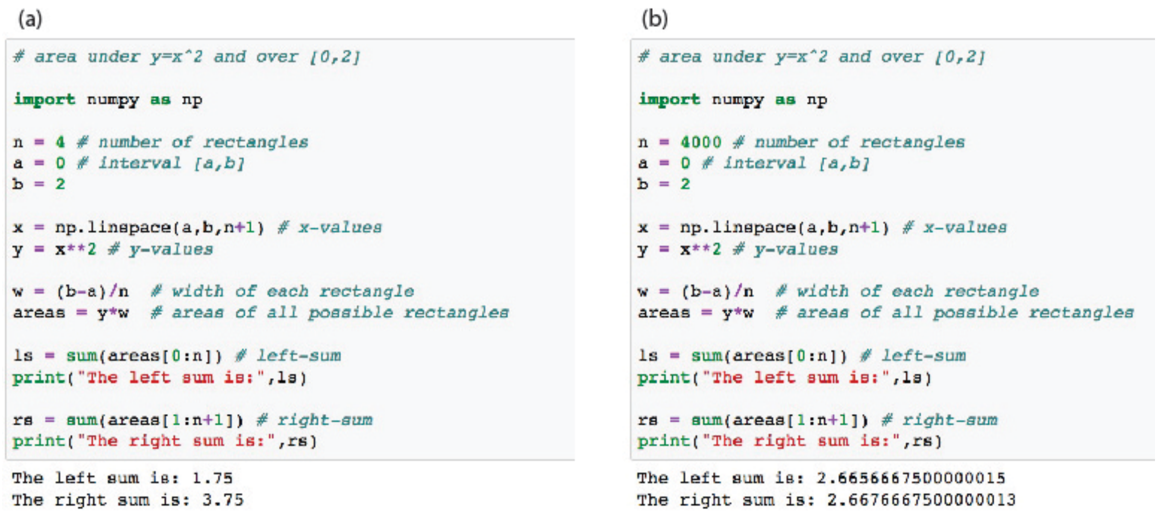
The left sum in Figure 15 is

$$\begin{aligned} L_4 &= \sum_{i=0}^{3} f(x_i)\Delta x \\ &= f(x_0)\Delta x + f(x_1)\Delta x + f(x_2)\Delta x + f(x_3)\Delta x \\ &= (0)(0.5) + (0.25)(0.5) + (1)(0.5) + (2.25)(0.5) \\ &= 1.75 \end{aligned}$$

The right sum in Figure 15 is

$$\begin{aligned} R_4 &= \sum_{i=1}^{4} f(x_i)\Delta x \\ &= f(x_1)\Delta x + f(x_2)\Delta x + f(x_3)\Delta x + f(x_4)\Delta x \\ &= (0.25)(0.5) + (1)(0.5) + (2.25)(0.5) + (4)(0.5) \\ &= 3.75 \end{aligned}$$

Reformulating a definite integral for computation requires that we first adopt a numerical approach to integration, that is, we approximate the value of the definite integral by using a Riemann sum with a finite number of rectangles n. In doing so, we represent the function $f(x)$ as a set of discrete values (discrete points). We then reformulate this problem for computation by assigning variables and parameters to quantities, and using a loop structure to compute the appropriate Riemann sum. These reformulations require a significant cognitive effort, as we are engaged with, and continuously switch between, abstract algebraic notions (functions as discrete objects, infinite summation), geometric representations (functions as graphs, regions bounded by curves, approximating rectangles) and numeric formulas and algorithms (calculating areas, summations, limits). It should be noted that when we work numerically and add together a finite number of rectangles, we generally obtain an approximation rather than the actual value of the definite integral; however, by combining this idea with the power of a computational representation, we can increase the number n until we have a sum as close as desired to the exact value of the definite integral.



(a)

```
# area under y=x^2 and over [0,2]

import numpy as np

n = 4 # number of rectangles
a = 0 # interval [a,b]
b = 2

x = np.linspace(a,b,n+1) # x-values
y = x**2 # y-values

w = (b-a)/n  # width of each rectangle
areas = y*w  # areas of all possible rectangles

ls = sum(areas[0:n]) # left-sum
print("The left sum is:",ls)

rs = sum(areas[1:n+1]) # right-sum
print("The right sum is:",rs)
```

```
The left sum is: 1.75
The right sum is: 3.75
```

(b)

```
# area under y=x^2 and over [0,2]

import numpy as np

n = 4000 # number of rectangles
a = 0 # interval [a,b]
b = 2

x = np.linspace(a,b,n+1) # x-values
y = x**2 # y-values

w = (b-a)/n  # width of each rectangle
areas = y*w  # areas of all possible rectangles

ls = sum(areas[0:n]) # left-sum
print("The left sum is:",ls)

rs = sum(areas[1:n+1]) # right-sum
print("The right sum is:",rs)
```

```
The left sum is: 2.6656667500000015
The right sum is: 2.6676667500000013
```

Figure 16: Python code for computing the left and right Riemann sums. (a) Using 4 rectangles (b) Using 4000 rectangles. Note that the command "sum" accomplishes the work of an entire loop, by adding the areas of the rectangles. Comparing with the output shown in (a), we see how, when 4000 rectangles are used (instead of 4), the two sums are very close to one another.

5.2.2 Affordances of Reformulation

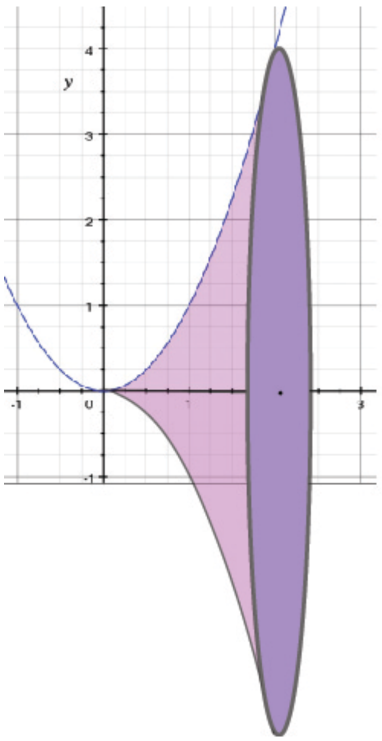
Students commented that thinking about how to reformulate their textbook problems and algorithms for computation (that is, thinking computationally), helped facilitate a more in-depth conceptual understanding of the underlying mathematical ideas. For example, students discovered they could generate multiple iterations of Euler's method effectively by using a loop structure in Python 3 (see Figure 12). They reported that the process of reformulating Euler's method for computation, that is, using computer code to define appropriate recursion relationships and using a loop to generate iterations of the solution, helped them to deeply understand the logic, structure, and algorithm (in both its algebraic and computational forms). Students stated that adopting different perspectives (and different representational systems) to analyze a problem, and comparing the complementary formulations of a solution algorithm, helped them develop a more comprehensive, intuitive, grounded understanding of the concepts.

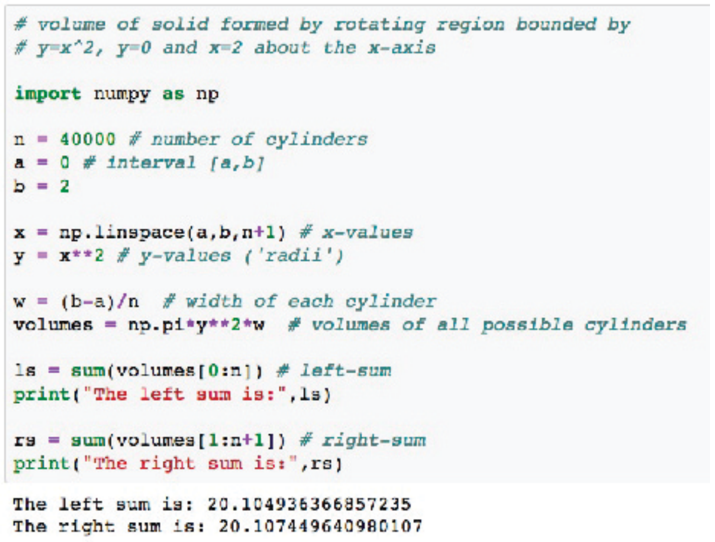
Students remarked that breaking a problem down into basic elements (sub-problems) in order to reformulate it for computation helped reduce the overall complexity of the problem and forced them to pay particular attention to all aspects of the task, as well as to the way in which these different aspects need to be put together. They noted that they were required to develop an in-depth understanding of the relationships between the quantities involved and the logic behind the solution algorithm in order to effectively reformulate the problem for computation. Students reported that the reformulation process revealed the simplicity underlying certain mathematical ideas, techniques, and algorithms. For example, while the difficulty of evaluating a definite integral algebraically varies greatly (in fact, many cannot be solved algebraically), reformulating integration for computation enables any proper[3] definite integral to be estimated by adding together the areas of approximating rectangles (a relatively simple task). Furthermore, students were surprised to discover that this simple idea could be readily extended to solve higher-level problems and applications, such as finding the volume of

[3] $\int_a^b f(x)dx$ is classified as a proper definite integral if the function $f(x)$ is continuous on the closed, finite interval $[a, b]$.

an irregular solid, using sums of volumes of cylinders, with a few minor modifications to their basic code (see Figure 17).

Moreover, using computational tools, students discovered a surprisingly simple, versatile mathematical approach to estimating unknown quantities (or, what could be considered a “big idea” in mathematics): begin with a simple numerical approximation, and then modify or adjust the approach (for instance by making it algorithmic, so that it can run in a loop) until this approximation becomes arbitrarily close to the actual value.





```
# volume of solid formed by rotating region bounded by
# y=x^2, y=0 and x=2 about the x-axis

import numpy as np

n = 40000 # number of cylinders
a = 0 # interval [a,b]
b = 2

x = np.linspace(a,b,n+1) # x-values
y = x**2 # y-values ('radii')

w = (b-a)/n  # width of each cylinder
volumes = np.pi*y**2*w  # volumes of all possible cylinders

ls = sum(volumes[0:n]) # left-sum
print("The left sum is:",ls)

rs = sum(volumes[1:n+1]) # right-sum
print("The right sum is:",rs)
```

```
The left sum is: 20.104936366857235
The right sum is: 20.107449640980107
```

Figure 17: Basic Riemann sum code from Figure 16 modified to estimate the volume of the solid obtained by rotating the region bounded by $f(x) = x^2$, $y = 0$, $x = 0$, and $x = 2$ about the x-axis.

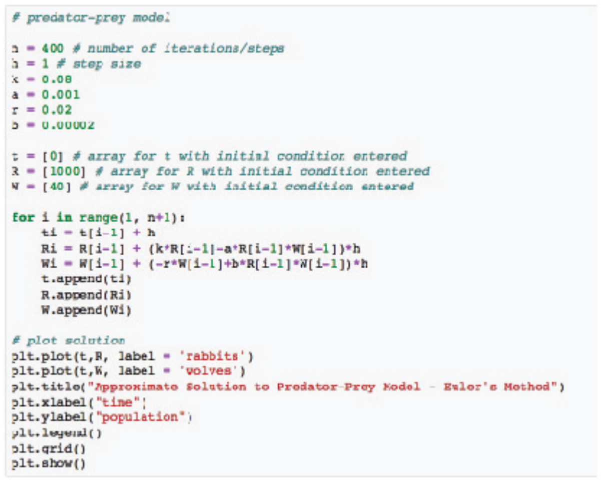
5.3 Reorganization

Adopting a new literacy has the potential to reorganize the intellectual landscape in profound ways, effectively rewriting the narrative of who gets to do what, and when. In other words, immersion in a new literacy and the ramifications of this immersion expand the range of what can be done, how it can be accomplished, and who is able to do it.

In Math 1LS3, exploring calculus concepts with computer code enabled students to effectively investigate meaningful, authentic, interdisciplinary applications, which were formerly inaccessible (and thus omitted from the course) due to overwhelming, technical complexities. This approach changed the traditional learning trajectory for our students and reorganized the intellectual domain of calculus, by engaging novice first-year students in activities typical for a graduate-level, research-based mathematics course. (Note that this illustrates the "low floor, high ceiling" affordance of computational thinking, as discussed by Gadanidis et al. (2016).)

Students attributed this achievement to the unique affordances accessible to them when they integrate computer coding into mathematical problem solving. For example, reformulating the problems to allow for a numerical approach, and remediating their investigations with computation (consequences of a new literacy) helped to significantly lessen the workload by removing numerous, repetitive, technical computations required when exploring complex problems. Students discovered that when represented computationally, theoretical (and often abstract) ideas can just as easily be applied to technically complex mathematical objects as they are to more basic cases.

For instance, using a computational model, students marveled at how straightforward it was to extend Euler's method to investigate solutions to systems of first-order differential equations, without noticeably increasing the demands on the computational aspects (such as the time Jupyter needed to complete the calculations). This enabled them to explore more complex models, such as the Susceptible-Exposed-Infected-Recovered model (SEIR-model), used to study the spread of the EBOLA virus during the recent epidemic in Africa, or the classical predator-prey model, which investigates the dynamics between two species interacting in a common habitat (see Figure 18).



```
# predator-prey model

n = 400 # number of iterations/steps
h = 1 # step size
k = 0.08
a = 0.001
r = 0.02
b = 0.00002

t = [0] # array for t with initial condition entered
R = [1000] # array for R with initial condition entered
W = [40] # array for W with initial condition entered

for i in range(1, n+1):
    ti = t[i-1] + h
    Ri = R[i-1] + (k*R[i-1]-a*R[i-1]*W[i-1])*h
    Wi = W[i-1] + (-r*W[i-1]+b*R[i-1]*W[i-1])*h
    t.append(ti)
    R.append(Ri)
    W.append(Wi)

# plot solution
plt.plot(t,R, label = 'rabbits')
plt.plot(t,W, label = 'wolves')
plt.title("Approximate Solution to Predator-Prey Model - Euler's Method")
plt.xlabel("time")
plt.ylabel("population")
plt.legend()
plt.grid()
plt.show()
```

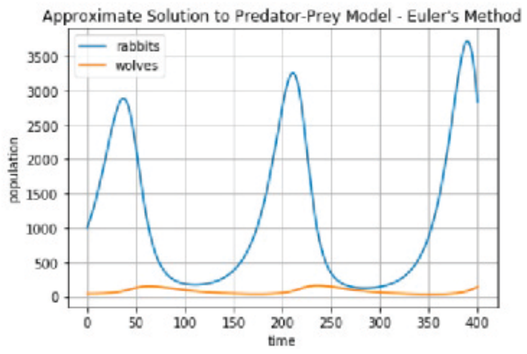


Figure 18: By modifying the base code for Euler's method, we can approximate solutions of a system of two differential equations. As previously, Euler's method is accomplished in one loop (left). The code outputs approximate solution curves for each of the functions (right).

Students reported that working on these authentic applications helped to increase the relevance and value of the material they were studying, which motivated them to further engage with their explorations and ask hypothetical questions, such as, "what would it look like if we mediated this particular model with computation and also explored it using our calculus concepts?" This allowed our students to modify their learning trajectories by diversifying their mathematical explorations in the ways that would be inaccessible using an algebraically mediated approach only.

In addition to increasing accessibility to authentic, interdisciplinary applications, remediating calculus concepts with computation provides an alternative approach to mathematical problem solving, which has the potential to support diverse learning styles. In particular, students noted that the coding activities were especially attractive to, and beneficial for, visual and kinesthetic learners, allowing them to directly interact with the concepts and receive immediate, dynamic, visual feedback. As well, students reported that using a computational representation of the models and algorithms enabled them to explore concepts and ideas in ways that were meaningful to them. Thus, it became evident that providing multiple avenues to access mathematical content broadens the range of learners who are able to successfully engage with undergraduate calculus.

5.4 Revitalization

diSessa (2018) explains that a new literacy has the potential to transform teaching and learning experiences, resulting in a revitalization of the learning ecology. As I discuss below, this revitalization is fundamentally connected to the principles of remediation, reformulation, and reorganization.

5.4.1 Learning

In Math 1LS3, we experienced a revitalization of our teaching and learning experiences when we integrated coding activities with our mathematical explorations. For instance, students reported that remediating calculus concepts with computation provided a fresh, modern approach to mathematical problem solving. They stated that this made the material feel more interesting, simulating and relevant, which overall increased their enjoyment of their learning.

The dynamic and interactive nature of the coding activities in Math 1LS3 offered students opportunities to explore, experiment and play with the mathematical concepts. They said that the coding activities opened up a creative space in mathematics that they had never experienced in other problem-solving situations, such as in a linear algebra course. Students reported that they enjoyed the flexibility of the opportunities available to them, and having options on problem-solving strategies was appealing and increased their interest. As well, the consistent and immediate feedback afforded by the coding activities helped them to shape and reinforce their understanding concurrent with their explorations, which students stated improved their confidence with their answers and overall conceptual understanding of the material.

Students remarked that analyzing mathematical models and algorithms using code provided differentiated learning opportunities, which supported a variety of learning styles. They felt free to experiment with the code in ways that were personally meaningful for them and didn’t stress about making mistakes, embracing trial and error as an important part of their learning process. This empowered those who felt they were unable to learn or fully understand mathematics within algebraic environments, and offered salient alternatives to traditional mathematical problem-solving strategies. Students felt they had more agency in their learning and experienced a greater feeling of satisfaction and accomplishment. As well, students noted that the coding activities stimulated peer collaborations, resulting in fruitful discussions and sharing of ideas.

Furthermore, students reported that the ability to directly apply calculus concepts to analyze authentic, contemporary problems (a consequence of the reorganization principle) effectively illustrated the value of the mathematical concepts they were learning. They explained that incorporating interdisciplinary applications made the material feel more interesting, simulating and relevant, which increased their enjoyment of their learning.

5.4.2 Teaching

An unexpected, but important, outcome was the revitalization of teaching experiences for instructors. This revitalization was most evident in the enriched capabilities afforded by computation, which dramatically expanded the range of interdisciplinary applications we could effectively incorporate into course material, and the capacity to investigate them, so that we could meaningfully, and authentically, engage with (and convincingly illustrate the value of) the mathematical material we were teaching.

For example, one of our first coding activities invited students to develop a program to apply Euler's approximation method n times to a first-order differential equation when given an initial condition. The obvious advantage in using computer technology to explore this iterative method is that n can be made very large without any extra human effort, which improves our estimation within any desired degree of accuracy. While this slightly improved my experience teaching Euler's method, I really became excited when I realized that I could introduce students to more complex models of systems of differential equations, where current research efforts in many branches of applied mathematics and life sciences are concentrated. There was literally no system of first-order differential equations that was off limits to us due to its complexity, and after typing out my code for our basic case in one cell, it was easy to copy and paste the code in a new cell, and make minor modifications so that it applied to anything I chose to explore. While in other math courses we talk about how to improve our estimation theoretically, it is rewarding and satisfying to actually demonstrate this improvement in a concrete way using computer code.

I felt that as students watched me do this spontaneously during lecture in less than a minute, I was giving them a realistic picture of how problems are explored outside of the classroom, while adding value to the mathematical material and showing its wide applicability when integrated with coding technology. The affordances of integrating coding technology into our teaching practice absolutely revitalized my enthusiasm for teaching (especially certain material that I have always perceived as "dry") by providing multiple ways to explore many of our traditional calculus topics.

5.5 The Acquisition of a New Literacy is a Massive Social and Cultural Accomplishment

diSessa (2018) defines a literacy as "a massive social/intellectual accomplishment of a culture or civilization, where many competing forces, over decades or centuries, eventually settle on a particular representational form for wide-spread learning, use, and subsequent value" (p. 7).

Remediating mathematical concepts with computation, and integrating computer technology to access unique computational thinking affordances, has played a role in mathematics education for several decades and was integral to Papert's (1980) innovative research using Logo (Gadanidis, 2018). However, this initiative did not achieve widespread attention until Jeanette Wing's (2006) influential paper inspired a resurgence in the interest of teaching computational thinking outside of a computer science context. Noss and Hoyles (1992) suggested that the reasons why computational thinking did not achieve a more prominent position in education alongside early initiatives revolved around certain social, cultural and pedagogical attitudes.

Today, with computer technology omnipresent and the growing widespread recognition of the value of computational thinking skills, the current social and cultural milieu is more conducive for advancing a computational literacy. Even so, incorporating computer labs into Math 1LS3 was initially met with some resistance, as students expressed their apprehension in using a computational representation system (in fact, in the first set of responses, many students reported that they didn't see the point of learning computer programming in a calculus course). This primary reaction was anticipated by diSessa (2018) and he states that "initial resistance and long periods of incubation are undoubtedly the norm" (p. 15) for any new literacy.

As the course progressed and students persisted in the coding activities, their comfort navigating the coding environment and their fluency in the programming language quickly improved. The prevalence of technology in our students' lives was likely the reason (at least in part) behind this accelerated familiarity with computational mathematics and coding tools. Needless to say, this is a very different environment from the one in which Papert introduced his ideas. As a result, students were increasingly able to communicate their ideas using computer code, in various effective and creative ways. Overtime, we (myself, other instructors, and our computer lab teaching assistant) noticed that students were relying less and less on the coding templates we provided, and instead, creating their own computational tools for representing, exploring, and solving problems in innovative ways, often moving their investigating above and beyond what was required in the original problem. Reflecting on their experience, one student said:

> *When I'm coding, I personally feel very engaged with the source material. It's as if this is my project, it's a problem and a journey, as I endeavour to solve it.*

By the end of the course, students' attitudes towards computer programming in mathematics changed dramatically as their programming skills had sufficiently developed and they could personally experience the power, versatility, and learning potential of combining computer programming with mathematics. This led to multiple requests for recommendations of other courses they could take which adopt a computational approach to mathematics (courses we are actively working on developing now!).

Within the Department of Mathematics and Statistics, there has been a significant interest in adopting our approach to integrating computer programming into other courses, without significantly changing the core mathematical content. While many courses currently use computer technology to supplement course material (e.g., Matlab, Maple, Excel), aside from computational mathematics and statistics courses, these courses do not officially teach a modern computer programming language, such as Python 3. Presumably, widely incorporating computer programming into other courses as a universal approach to problem solving would help improve students' proficiency with the language and coding environment, and further establish its role as a versatile problem-solving strategy.

I have collaborated with several faculty members in other departments (e.g., Department of Physics at McMaster University) and universities (e.g., Mathematics at University of Toronto Mississauga) to offer guidance and resources for integrating computational thinking into their current courses. Additionally, I discussed our labs with a colleague from University of Waterloo, who is interested in offering Python labs to their students.

The keen, growing interest I've personally experienced toward incorporating computer programming into a wide variety of courses outside of computer science illustrates the recognition of the value and potential of a computational representational system. Furthermore, the "social spread" of this endeavor that I have witnessed following the success of my pilot semester (e.g., within our course, department, university and beyond) provides evidence of a budding computational literacy in the sense that diSessa (2018) conceptualizes it.

5.6 Limitations of a Computational Approach

As diSessa (2018) explains, all representational systems have their own unique affordances and limitations. In Math 1LS3, remediating integration with computation resulted in a powerful and versatile numerical approach to integration; however, approximating a definite integral using a Riemann sum produces only an estimate of the definite integral, which is sufficient in most applied mathematical research but is still theoretically different from evaluating a definite integral. While students reported that exploring Riemann sums using code enriched their conceptual understanding of the technique and its relationship to definite integrals, there was little evidence to suggest that this enhanced understanding of integration improved their ability to evaluate integrals algebraically, a process many students in Math 1LS3 still experience difficulty with.

A natural limitation of a computational approach is the mathematics content itself—certain topics and ideas cannot be investigated (in their completeness, or at all) by coding. For instance, whereas computing a finite sum is a straightforward exercise in Python, no code can prove convergence or divergence for an infinite sum (infinite series). As well, calculating a table of values for a function to determine its limit could lead to erroneous conclusions. By extension, coding cannot prove that a given equilibrium of a dynamical system is stable, as it can compute only a finite number of steps. Thus, mathematical results and ideas that require inductive reasoning, that is, making and proving generalizations based on specific examples, cannot be approached using computational tools. Of course, coding can provide some evidence that a generalization might be true, but it cannot prove it to be true.

All coding languages have a demanding and rigorous syntax, and even an extra space in the wrong place could generate an error and prevent the code from functioning as desired. This specificity was a common source of frustration for many students throughout the semester (not to mention the time they needed to figure out the source of the problems), and this technical limitation is a common issue in computer programming in general.

A further limitation of a computational approach is due to its nature (i.e., inability to "think" beyond the code given), which, coupled with students' (mis)beliefs about what it actually does, leads to erroneous answers. For instance, students discovered that just because their code runs without error messages and returns an answer, does not mean that its output is a (correct) solution to the problem they were trying to solve. For instance, misplaced parentheses could change the formula for a function that is analyzed, or an inadequate number of steps could lead to a poor approximation of a definite integral. Python has no way of reading users' minds to guess their intentions—it does exactly what the code tells it to do, nothing more or nothing less. In other words, it is unable to warn a student that there is a mathematical inaccuracy with their code.

5.7 Summary

Assuming diSessa’s (2018) theoretical perspective enabled me to conduct a deeper analysis of my results and situate them within the framework of a new computational literacy. As evidenced by their reflective responses, when Math 1LS3 students reformulated differential and integral calculus concepts in order to remediate them with computer code, their learning experiences were transformed, resulting in a reorganization of the intellectual landscape and revitalization of their learning ecology. These four principles reflect the “massive social and cultural accomplishment” (diSessa, 2018, p. 25) of establishing a new, in this case computational, literacy.

Chapter 6

6 Conclusion

Computational thinking (and the set of computational tools that facilitate it) has received acclaim for its potential to support, enrich and innovate problem-solving activities in a wide variety of contexts. A computational thinking approach in mathematics offers a powerful set of affordances stemming from both the underlying processes—in particular, abstraction and automation—and from appropriately designed coding activities, which can further enhance not only problem solving, but also mathematical reasoning, understanding, and learning in general. When effectively integrated into educational activities, computational thinking has the potential to provide unique, transformative learning experiences to students. For instance, it can enrich and expand the means and tools available to students in their mathematical explorations, learning of concepts and problem-solving activities.

While there is a significant body of literature on the theoretical aspects of computational thinking in education, there is a relatively large gap in the literature providing practical, specific guidance for its integration into various subject areas, as well as a critical, evidence-based analysis of such integration efforts.

6.1 Current Study

This research project investigated an approach to integrating computational thinking into a first-year, undergraduate calculus course designed specifically for life sciences students. In collaboration with the course coordinator, I developed a set of mathematical coding activities (organized into four computer labs) to supplement and enhance mathematical problem solving, as well as promote a richer understanding of the course content, while taking advantage of the unique affordances computational thinking can offer to enhance educational experiences. My goal was not just to integrate technology into our classroom, but to enrich and transform the ways students see mathematics, and to modernize the teaching of mathematics at the undergraduate level.

A series of questions and prompts that followed each of the four computer labs invited students to reflect on their experiences of combining mathematics with coding. Each survey was designed to solicit insights into changes in students' conceptual understanding, which resulted from interacting with the mathematical coding activities, as well as to inquire about students' affective responses to this integrated learning experience.

Students' responses were collected and analyzed, first using a qualitative content analysis to organize the data into categories (and later, overarching themes), and then using diSessa's (2018) literacy framework to help theorize about the results obtained, and to achieve a "big picture" view of computational thinking as a literacy.

6.2 Results

My content analysis revealed three central themes within students' responses: modified perceptions of mathematics, enhanced mathematics learning experiences, and unique coding affordances.

Students reported that the engagement with coding activities within their calculus course changed their perceptions of what mathematics is in several ways. They had opportunities to effectively explore and analyze authentic applications in the life sciences, which provided a broader, more representative perspective of the field of (applied) mathematics. Students noted that the ability to combine standard calculus tools with coding effectively illustrated the relevance and value of the mathematical concepts. For example, because it is initially presented as an abstract concept, students often do not appreciate the importance of difference quotients when calculating derivatives; however, they soon realize that when dealing with discrete data (as often is the case in real-life contexts), using differentiation rules is not an option!

Students described how exploring calculus concepts with computer code enriched and transformed their mathematics learning experiences throughout the semester. They remarked that using coding for their mathematical explorations facilitated a dynamic, interactive learning experience, which motivated them to be more actively engaged with the material, compared with traditional, non-coding versions of the (same, or similar) problems. They reported that the opportunities to explore, experiment, and play with the concepts—combined with the immediate feedback and dynamical visualizations that accompanied running code—promoted a deeper, more comprehensive understanding of the mathematical content and a greater enjoyment of the problem-solving process. Additionally, students noted that the coding activities accommodated, in their words, alternative learning styles more effectively than traditional, paper-and-pencil strategies, and invited multiple approaches and flexibility during the problem-solving process. Many remarked how this alternative approach encouraged meaningful peer collaborations and creative problem-solving strategies, two features students noted were typically lacking in traditional mathematics courses.

Students observed that several coding affordances enabled them to explore calculus concepts in novel ways. For example, improved technical capabilities afforded by the computer technology facilitated efficient, accurate calculations in even the most complex instances, motivating students to apply their theoretical knowledge to solve complex, authentic, real-world problems using standard undergraduate calculus concepts. As well, students noted that exploring concepts with code helped give the abstract theoretical material a “tangible feel.” This helped them make important connections between the theory and practice—a well-known challenge many students encounter in mathematics courses. Exploring calculus concepts using code (that had to be generated) promoted a greater understanding of the theoretical concepts, and was more rewarding, compared to using prepackaged applications, since coding gave them full control over the entire problem-solving process. Students also remarked that the physical process of coding (that is, automation and abstraction) provided additional benefits, such as reducing the complexity of a problem by breaking it down into its basic elements in order to reformulate it for computation.

6.3 Analysis

Analyzing my data using diSessa’s (2018) literacy framework enabled me to adopt a different frame of reference, and hence an alternative perspective on my data, and helped situate my research within other initiatives in education.

diSessa (2018) explains that remediating concepts, problems, and processes with a new representational system affords unique opportunities to engage with ideas in novel ways. In Math 1LS3, students found that using a computationally mediated approach enabled them to effectively incorporate computer technology into their investigations, which offered significant technical advantages. They remarked that exploring models, concepts, techniques, and algorithms with computer code optimized their problem-solving activities, which helped them to expand their explorations beyond previous (technical) constraints. Students noticed that they were able to focus more on developing their conceptual understanding and overall perspective on the underlying ideas and concepts when their mental energy was not expended on lengthy, complex, and repetitive calculations.

Reformulating all concepts, problems, and processes related to an investigation often involves a significant cognitive shift, but has the potential to reveal cognitive simplicities in the underlying concepts. In Math 1LS3, reformulating integration for computation naturally revealed that the area of a bounded region could be estimated to within any degree of accuracy using a sufficient number of approximating rectangles. Further extending this idea, students discovered that volumes of irregular solids could be adequately approximated using approximating cylinders.

As diSessa's reorganization principle projected, integrating coding activities into our undergraduate calculus course reorganized the intellectual terrain in profound ways. For example, remediating calculus concepts with computation enabled first-year undergraduate students to profitably engage with graduate (and research) level mathematics, within the first few weeks of classes. Furthermore, alternative approaches (in this case, a computational approach) helped accommodate a broader group of students, thus increasing the number of students who can successfully engage with calculus concepts. These two outcomes changed the predetermined learning trajectory for students and rewrote the narrative of who is able to effectively learn calculus.

diSessa (2018) explains that a new literacy has the potential to refresh and invigorate the teaching and learning ecology, which we witnessed extensively throughout the semester.

Students noted multiple affordances of computational tools (e.g., dynamic modelling in an interactive development environment, rich visualizations, efficient calculations) that dramatically transformed their learning experiences in the course. For instructors, one of the most noteworthy contributors to the revitalization of teaching was the ability to effectively analyze authentic models and applications—thus demonstrating the value and relevance of the mathematical concepts—without the constraints of complex, tedious calculations.

diSessa (2018) describes a literacy-scaled achievement as a massive social and cultural endeavour. Witnessing the rapid "social spread" of this initiative (beginning within Math 1LS3, and then expanding to our department, Faculty of Science, and beyond), completed the final piece of diSessa's "five principles of a literacy" puzzle, and provided sufficient evidence that the results of this initiative indicate, at the very least, a "budding computational literacy" (diSessa, 2018, p. 8).

6.4 Limitations of a Computational Representation

The largest challenges reported by students stemmed from the technical side, that is, from the difficulties with the particular representational system. (Given diSessa's (2018) proclamation that every representational system has its weaknesses, this is not at all surprising.) Since computer languages are highly specific and demanding in regards to their syntax, something as simple as an extra space in the wrong place could cause the code not to function as desired. Students found this frustrating and suggested that they should be explicitly taught the coding language first, if they were expected to use it effectively. While we embedded sufficient sample code, explanations, and illustrative examples to complete each lab, we did not attempt to comprehensively teach a coding language. Instead, students were encouraged to learn additional features of Python 3 on an "as needed" basis and to seek additional help by using the many coding resources available online.

Students reported that searching for appropriate online resources was frustrating and time-consuming, and that the information they found was often not directly applicable to the task they were working on. They requested a video tutorial, created specifically for

Math 1LS3, to guide them through the basics of coding, Jupyter notebooks, and Python 3. To address their concerns, I created a thirty-minute video tutorial titled, *Getting Started in Jupyter Notebook* (https://www.youtube.com/watch?v=dEWsl4OUJ_c&t=1199s), which introduced students to the coding platform and basic concepts, strategies, and techniques needed for their labs. While I didn't feel that an additional YouTube video was necessary, students did find it quite useful, and reported that they felt their voice was heard. Furthermore, as I suggest below, initiatives coming from within a discipline—in this case, a Python 3 video tutorial created by a Math 1LS3 instructor—may be better received than a generic video produced elsewhere.

A significant number of students reported feeling more overwhelmed than inspired or excited at the prospect of learning computer programming in addition to the standard calculus content. They said that they spent too much time on coding and not enough time on algebraic techniques, which still form the greater part of our assessments and are (at present) more transferable to upper-year mathematics courses.

In reformulating mathematics problems for computation, that is, when switching from an algebraic model to a computational representation, we varied the theoretical content we aimed to explore (which is a natural consequence of reformulation, as described by diSessa (2018)); in particular, we reformulated continuous functions into a numeric form by representing them as a discrete set of points. Consequently, all calculus tools applied to this array of values were necessarily approximations of their continuous, theoretical counterparts. While many students found that this enriched and broadened their perspective of the concepts, and of mathematical modelling in general, some found it confusing, overwhelming, and reported that they had difficulty connecting the computational version to the original algebraic formulation.

We offered options to help mediate any discomfort or frustration students might feel with this new teaching and learning method. For example, numerous support structures were offered: all three instructors held several office hours each week and encouraged students to bring their laptops. In addition to our usual teaching assistants, we hired a "lab TA" who held five office hours each week to assist students with computer labs. Despite these efforts, there were students who were extremely resistant to learning computer programming in any capacity. To accommodate these students, we modified our course policy so that the term work grade would be calculated using the best three out of four assessments: three term tests and the set of computer labs, which collectively counted as the fourth assessment. While we strongly suggested that all students attempt the computer labs, we also explained that the labs are not mandatory. If a student chose not to complete the computer labs, then their grade would be based on their three term tests. This decision reflected the fact that coding is not necessarily for everyone and that no single representation can claim to be universally superior, for everyone, in all situations. As diSessa (2018) stresses, all representational systems possess "distinctive and critical strengths, but also limitations and blind spots, and, thus, a possible complementarity with other forms of representation" (p. 7).

6.5 Limitations of the Current Study

The findings and conclusions of this study were based on students' subjective, self-reported responses, which are vulnerable to several sources of bias. For example, we need to trust that students have sufficient self-awareness to recognize and accurately report their experiences, and that their intentions were to respond truthfully. As well, the sample of students was biased in that the majority are enrolled in the Life Sciences program. While this potentially limits the transferability of my results (at least until further research is conducted), the sample size was large enough that I can confidently say my findings represent the views and experiences of life sciences students.

In my extensive review of the literature, I could not find a study similar to mine, and so I was unable to directly compare my results to other studies. As well, my research was conducted over the course of one semester, and so, only once. The credibility and dependability of my findings would improve if this research was conducted in several semesters, and the results were replicated.

6.6 Suggestions for Future Research

Broadly speaking, my research contributes to a growing body of literature aiming to access the best that computational thinking has to offer and to effectively integrate it in areas where it may enrich and enhance problem solving. In particular, the results obtained through this study contribute to the ongoing, SSHRC-funded research project, *Computational Thinking in Mathematics Education* (http://ctmath.ca/), by offering an analysis of a practical approach to integrating computational thinking, in a meaningful capacity, into a large, undergraduate calculus course.

Going forward, we will continue to incorporate computer labs in Math 1LS3, keeping what we learned was beneficial from the pilot semester, and further developing areas which need to be improved. We also plan to offer coding labs for the second half of the course, Math 1LT3: Calculus II for the Life Sciences. In fact, students who successfully completed Math 1LS3 in the fall of 2018 were disappointed to learn that computer labs were not a formal component in Math 1LT3 during the winter 2019 semester, which we interpreted as a testament to the success of this initiative! Currently, we are generating a collection of computational modelling activities for Math 1LT3, and we plan to continue to expand this line of work, using students' feedback, to create an exciting stream of computational calculus at McMaster University. Needless to say, assessing the efficacy of this implementation is an ongoing research objective.

As suggested by diSessa (2018), and now from my own personal experiences, I believe that future efforts to expand computational thinking into all disciplines will be most successful if they arise from *within* a certain discipline. This will minimize the emphasis on computer science and ensure that the true computational thinking principles and transferable skills—not just technical programming skills—are being realized. As well, if we are to look beyond computational thinking and consider the potential achievement of a true computational literacy, as defined by diSessa (2018), then the efforts must come from all areas in society since "the professional pursuit of understanding or creating a literacy—or anything that has similarly broad aspirations—cannot belong in any substantial degree to one of the standard professional disciplines" (p. 18).

My current research project has received positive attention from both within my department and beyond, which opens doors for future research collaborations. In particular, I am interested to investigate (jointly with colleagues from the respective mathematics departments) the integration of computational thinking at the University of Waterloo and the University of Toronto Mississauga, in the hope of replicating my results, and expanding my study.

Of course, the integration of computational thinking at McMaster University requires further research scrutiny, to support some of my beliefs with additional evidence, and to further to strengthen existing evidence. For instance, I believe that as coding, (and, more important, computational thinking) are incorporated and reinforced in a variety of contexts, students will perceive them as more versatile, valuable and relevant, and consequently be better motivated to invest time into learning the basics of programming. As the technical challenges lessen with increased exposure, students will likely feel more confident with programming in general, and their initial resistance to coding activities, as students in the fall 2018 semester of Math 1LS3 experienced, may decrease.

The video tutorial resource students requested was very well received and students requested that shorter videos be created and posted online for individual topics, such as using loops, plotting functions, etc. In the future, we plan to recruit students to share their own approaches to mathematical problems using screen recordings and feature some of these videos on our Math 1LS3 YouTube channel. Studying how creating and using these short(er) videos affect students' learning is another important research direction.

In the pilot semester, the coding problems and applications were presented in a prescriptive (i.e., scaffolded) manner—students were encouraged to explore, but under controlled conditions. In future semesters, with appropriate and sufficient support and resources in place (such as a collection of short, student-generated, single topic, course-specific video tutorials available on a course YouTube channel), we will strive to engage students in the process of reformulating problems for computation and remediating them with code, with minimal scaffolding. Engaging students in the act of reformulating mathematical problems and asking them to create their own algorithms to generate solutions, requires a stronger commitment but offers the greatest potential to experience the maximum benefits of computational thinking, while providing a strong sense of student agency and control throughout the construction of knowledge and learning.

Our future efforts at improving our computer labs, and integration of computational thinking in general, will require strong theoretical support—hence, there will be a strong demand, and plenty of opportunities for further research, for myself, and for my graduate students. For instance, I could conduct observational-type research to determine, on a "microscopic level," how individual students work with, and learn from, the coding activities. Additional strength and significance of this study can be achieved by conducting complementary research, for example, by investigating computational modelling within an undergraduate physics course or a secondary-level mathematics course.

6.7 Summary

Computational thinking is used to describe a set of thinking or problem-solving strategies, which parallel, and are inspired by, certain computer programming processes and techniques. Research has suggested that computational thinking and related activities, such as computer coding, have the potential to provide a useful and powerful problem-solving framework, which can (in some instances) extend into non-computer science domains.

Computational thinking has innovated, transformed, and revitalized teaching and learning experiences in profound ways. For example, diSessa (2018) demonstrated how reformulating concepts of motion and remediating them with computation revitalized learning experiences and reorganized the intellectual domain for sixth grade students. Despite encouraging evidence that indicates computational thinking could be a valuable new literacy, it has not yet been effectively integrated into many subjects to augment problem-solving activities.

For my doctoral research project, I investigated an approach to integrating computational thinking into an undergraduate calculus course. Working together with the course coordinator, I designed a set of computer programming activities to complement and enrich our calculus topics, and incorporated them into our course activities. I collected and analyzed students' reflections on the activities, which provided valuable insights into the cognitive and affective changes that occur when calculus is reformulated and remediated with code.

My research suggests that students' conceptual understanding and affective experiences were dramatically transformed through the integration of coding and calculus. This integration revitalized their learning experiences, changed their perception of the field of mathematics, and offered unique new opportunities to dynamically interact with the theoretical ideas. While students did experience some frustration with coding (all representations have natural limitations), the issues were largely technical and our future efforts will improve the resources students need to mitigate such issues.

In future semesters, I plan to modify the coding activities so that the problem-solving scaffolding is minimal, thus allowing students maximum opportunities to benefit from computational thinking. We will integrate Python 3 coding activities into other courses within the Mathematics and Statistics Department, and support other disciplines in their efforts to do the same. As computational thinking is effectively integrated into all subjects, from within each subject, students will likely perceive computational thinking as a powerful, useful, relevant, and highly applicable, transferable skill.

References

Aho, A. V. (2012). Computation and computational thinking. *Computer Journal, 55*(7), 832-835.

Althaus, C. L. (2014). Estimating the reproduction number of Ebola virus (EBOV) during the 2014 outbreak in West Africa. *PLoS Currents Outbreaks, 6.* doi:10.1371/currents.outbreaks.91afb5e0f279e7f29e7056095255b288.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48-54.

Berland, M., & Lee, V. R. (2011). Collaborative strategic board games as a site for distributed computational thinking. *International Journal of Game-Based Learning, 1*(2), 65-81.

Berry, M. (2013). *Computing in the national curriculum: A guide for primary teachers* [PDF file]. Retrieved from http://www.computingatschool.org.uk/data/uploads/CASPrimaryComputing.pdf

Birmingham, P. & Wilkinson, D. (2003). *Using research instruments: A guide for researchers.* Routledge.

Blum, W. & Borromeo Ferri, R. (2009). Mathematical modelling: Can it be taught and learnt? *Journal of Mathematical Modelling and Application. 1*(1), 45-58.

Boaler, J., Chen, L., Williams, C., & Cordero, M. (2016). *Seeing as understanding: The importance of visual mathematics for our brain and learning.* Retrieved from https://www.youcubed.org/seeing-understanding-importance-visual-mathematics-brain-learning/

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., Engelhardt, K. (2016). *Developing computational thinking in compulsory education – Implications for policy and practice.* EUR 28295 EN. doi:10.2791/792158

Borba, M. C., & Villarreal, M. (2005). *Humans-with-media and reorganization of mathematical thinking: Information and communication technologies, modeling, experimentation and visualization.* New York: Springer.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*(2), 67-69.

Burton, L. (1999). The practices of mathematicians: What do they tell us about coming to know mathematics? *Educational Studies in Mathematics, 37*, 121-143.

Clements, E. & Lovric, M. (2018). Chapter 3: Interdisciplinary approaches motivate and enrich teaching and learning mathematics. In Sibbald, T. (Ed.) *Teaching Interdisciplinary Mathematics (Learning to Teach, Teaching to Learn).* University of Illinois: Common Ground Publishing.

Cohen, L, Manion, L., & Morrison, K. (2007). Chapter 22: Approaches to qualitative data analysis. In *Research methods in education* (pp. 461-474). London, England & New York, NY: Routledge.

Computer Science Teachers Association. (2011). *K-12 computer science standards.* Retrieved from http://csta.acm.org/Curriculum/sub/K12Standards.html

Curzon, P., Black, J., Meagher, L. R., & McOwan, P. (2009). Enthusing students about computer science. *Proceedings of Informatics Education Europe IV*, 73-80.

Czerkawski, B. C., & Lyman, E. W. (2015). Exploring issues about computational thinking in higher education. *TechTrends, 59*(2), 57-65.

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM, 60*(6), 33-39.

diSessa, A. (2018). Computational literacy and "The Big Picture" concerning computers. *Mathematics Education, Mathematical Thinking and Learning, 20*(1), 3-31. doi: 10.1080/10986065.2018.1403544

Dodig-Crnkovic, G. (2011). Significance of models of computation, from Turing model to natural computation. *Minds & Machines, 21*(2), 301-322. doi:10.1007/s11023-011-9235-1.

Furber, S. (2012). Shut down or restart? The way forward for computing in UK schools, *Technical report*, The Royal Society. Retrieved from https://royalsociety.org/topics-policy/projects/computing-in-schools/report/

Gadanidis, G. (2015). Coding as a Trojan Horse for mathematics education reform. *Journal of Computers in Mathematics and Science Teaching, 34*(2), 155-173.

Gadanidis, G., Clements, E., & Yiu, C. (2018). Group theory, computational thinking, and young mathematicians. *Mathematical Thinking and Learning, 20*(1), 32-53. doi: 10.1080/10986065.2018.1403542

Gadanidis, G., Hughes, J., Minniti, L., & White, B. J. G. (2016). Computational thinking, grade 1 students and the binomial theorem. *Digital Experiences in Math Education*. doi:10.1007/s40751-016-0019-3

Gefter, A. (2016). The evolutionary argument against reality. *Quanta Magazine*. Retrieved from https://www.quantamagazine.org/20160421-the-evolutionary-argument-against-reality/

Grover, S. & Pea, R. (2013). Computational thinking in K-12. A review of the state of the field. *Educational Researcher, 42*(1), 38-43.

Guzdial, M. (2008). Paving the way for computational thinking. *Communications of the ACM, 51*(8), 25-27.

Hazzan, O. (1999). Reducing abstraction level when learning abstract algebra concepts. *Educational Studies in Mathematics, 40*, 71–90.

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads 1*(2), 4-7.

Henderson, P. B., Cortina, T. J., Hazzan, O., & Wing, J. M. (2007). Computational thinking. In *Proceedings of the 38th ACM SIGCSE Technical Symposium on*

Computer Science Education (SIGCSE '07), 195–196. New York, NY: ACM Press.

Hsieh, H. F. & Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative Health Research, 15*(9), 1277-1288. doi: 10.1177/1049732305276687

Hycner, R.H. (1985). Some guidelines for the phenomenological analysis of interview data. *Human Studies, 8*, 279-303. https://doi.org/10.1007/BF00142995

Kajander, A. & Lovric, M. (2009). Mathematics textbooks and their potential role in supporting misconceptions. *International Journal of Mathematical Education in Science and Technology, 40*(2), 173-181. doi: 10.1080/00207390701691558

King, K., Hillel, J., & Artigue, M. (2001). Technology – A working group report. In D. Holton (ed.) *The Teaching and Learning of Mathematics at University Level: An ICMI Study* (pp. 349-356). Dordrecht: Kluwer Academic Publishers.

Kolodziej, M. (2017). *Computational thinking in curriculum for higher education.* (Publication No. 10285666) [Doctoral dissertation, Pepperdine University]. ProQuest Dissertations.

Krippendorff, K. (2004). *Content analysis: An introduction to its methodology.* Thousand Oaks, CA: Sage Publications.

Kules, B. (2016), Computational thinking is critical thinking: Connecting to university discourse, goals, and learning outcomes. *Proceedings of the Association for Information Science and Technology, 53*(1), 1-6. doi:10.1002/pra2.2016.14505301092

Melfi, T. (2016). *Hidden figures* [Film]. Fox 2000 Pictures, Chernin Entertainment, Levantine Films.

Lewandowski, G., Bouvier, D., Chen, T. Y., McCartney, R., Sanders, K., Simon, B., & Vandegrift, T. (2010). Commonsense understanding of concurrency: Computing students and concert tickets. *Communications of the ACM, 53*(7), 60-70.

Lincoln, Y.S. & Guba, E.G. (1985). *Naturalistic Inquiry.* Newbury Park, CA: Sage Publications.

Lincoln, Y.S., Lynham, A., & Guba, E. G. (2011). Paradigmatic (model) controversies, contradictions, and emerging (developing) confluences (convergences), revisited. In N. Denzin & Y. Lincoln (Eds.), *The Sage handbook of qualitative research*, (4th ed., pp. 97-128). Thousand Oaks, CA: Sage.

Lye, S.Y. and Koh, J. H. L (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51-61.

Marshall, N. & Buteau, C. (2014). Learning mathematics by designing, programming, and investigating with interactive, dynamic computer-based objects. *International Journal of Technology in Mathematics Education, 71*(2), 49-64.

Mayring, P. (2000). Qualitative content analysis. *Forum: Qualitative Social Research, 1*(2). Retrieved from http://www.qualitative-research.net/fqs

Mohaghegh, M., & McCauley, M. (2016). Computational thinking: The skill set of the 21st century. *International Journal of Computer Science and Information Technologies (IJCSIT), 7*(3), 1524-1530.

Molnar, B. A. (n.d.). Computers in education: A brief history. *THE Journal.* Retrieved November 15, 2016, from https://thejournal.com/articles/1997/06/01/computers-in-education-a-brief-history.aspx

National Research Council. (2010). *Report of a workshop on the scope and nature of computational thinking.* Washington, DC: The National Academies Press. Retrieved from https://www.nap.edu/catalog/12840/report-of-a-workshop-on-the-scope-and-nature-of-computational-thinking

Noss, R. & Hoyles, C. (1992). Looking back and looking forward. In R.Noss & C.Hoyles (Eds.), *Learning mathematics and Logo* (pp. 431-468). MIT Press.

NVivo (n.d.). [Software]. Retrieved from https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York, NY: Basic Books.

Papert, S. (1993). *The children's machine. Rethinking school in the age of the computer.* New York, NY: Basic Books.

Pearson, K. (2009). From a usable past to a collaborative future: African American culture in the age of computational thinking. *Black History Bulletin, 72*(1), 41-44.

Pepper, R. E., Chasteen, S. V., Pollock, S. J., & Perkins, K. K. (2012). Observations on student difficulties with mathematics in upper-division electricity and magnetism. *Physical Review Special Topics-Physics Education Research, 8*(1), 010111. Retrieved from https://doi.org/10.1103/PhysRevSTPER.8.010111

Pesonen, M. E., & 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.

Resnick, M., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67.

Rukwaru, M. (2015). *Social research methods: A complete guide.* Meru: Eureka Publishers.

Sanford, J. F., & Naidu, J. T. (2016). Computational thinking concepts for grade school. *Contemporary Issues in Education Research (Online), 9*(1), 23-32. Retrieved from https://www.lib.uwo.ca/cgi-bin/ezpauthn.cgi?url=http://search.proquest.com/docview/1757523533?accountid=15115

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014). Exploring the science framework and the NGSS: Computational thinking in elementary school classrooms. *Science and Children, 52*(3), 10-15. Retrieved from https://www.lib.uwo.ca/cgi-bin/ezpauthn.cgi?url=http://search.proquest.com/docview/1627727555?accountid=15115

Soh, L-K., Samal, A., Scott, S., Ramsay, S., Moriyama, E., Meyer, G., Moore, B., T.G. Thomas & Shell, D. F. (2009). Renaissance computing: An initiative for promoting student participation in computing. *SIGCSE Bulletin, 41*(1), 59-63.

Stake, R. (2005). Qualitative case studies. In N. Denzin & Y. Lincoln (Eds.), *The Sage handbook of qualitative research,* (3rd ed., pp. 447-466). Thousand Oaks, CA: Sage.

Stewart, J. (2012). *Calculus: Early transcendentals* (7th ed.). Belmont, CA: Brooks/Cole.

Stillman, G. (2015). Applications and modelling research in secondary classrooms; What have we learnt? In S. J. Cho (Ed.), *Selected regular lectures from the 12th International Congress on Mathematics Education* (pp. 771-790). Cham, Switzerland: Springer.

Swaid, S. I. (2015). Bringing computational thinking to STEM education. *Procedia Manufacturing, 3*, 3657-3662.

Tall, D. & Association of Teachers of Mathematics (1987). *Understanding the calculus.* Derby, England: Association of Teachers of Mathematics.

Tesch, R. (1990). *Qualitative research: Analysis types and software tools.* Bristol, PA: Falmer.

Vygotsky, L.S. (1978). *Mind in society.* Cambridge, MA: Harvard University Press.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal for Science Education and Technology, 25*, 127-147.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society, A, 366*(1881), 3717-3725.

Wing, J. M. (2010). *Computational thinking: What and why?* Retrieved from http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Wolfram, S. (2016, September 7). How to teach computational thinking [Blog post]. Retrieved from http://blog.wolfram.com/2016/09/07/how-to-teach-computational-thinking/

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., and Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. *ACM Transactions on Computing Education 14*(1), 5:1-5:16.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S. & Korb, J. T. (2011). Introducing computational thinking in education courses. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education,* 465-470.

Yin, R. K. (2009). Introduction: How to know whether and when to use case studies as a research method. In *Case study research* (pp. 3-23). Los Angeles, CA: Sage.

Appendices

Appendix A: Letter of Information

Letter of Information

Project Title: Computational Thinking in Tertiary Mathematics Education
Principal Investigator: Dr. George Gadanidis, Faculty of Education, Western University
Co-Investigator: Erin Clements, Math 1LS3 Instructor, McMaster University

1. Invitation to Participate

You are being invited to participate in this research study on computational thinking in tertiary mathematics education because you are currently enrolled in an undergraduate mathematics course (Math 1LS3) at McMaster, which incorporates coding activities into instruction, learning and assessment practices.

2. Purpose of this Study

The purpose of this study is to gather feedback on students' experiences with the coding activities offered in Math 1LS3 (Calculus I for the Life Sciences). In particular, we would like to know how your conceptual understanding of abstract mathematical concepts is affected by engaging in coding activities and your subjective experiences with this engagement.

We are collecting data from you for two purposes: to improve present, and if successful, subsequent offerings of Math 1LS3, and to include these research findings in a doctoral thesis, *Computational Thinking in Tertiary Mathematics Education*.

3. Study Procedures

In this study, we would like to analyze your responses to the conceptual, procedural, and narrative questions posed in the coding labs. We would like to ask for your permission to use this data so that we can obtain an accurate and complete sample (i.e., large enough to be statistically significant). Your responses will give us valuable information about how your conceptual understanding of abstract mathematical concepts is affected by engaging in problem-solving activities which incorporate computational thinking as well as how you perceive these experiences.

We will ask for your consent (to use your responses) at the end of each lab. If you initially agree to be part of the study, but then change your mind, you can withdraw at any time up until 31 January 2019 by sending an email to the research assistant, Reihaneh Jamalifar [redacted]

Your participation, or withdrawal from participation, will not, in any way, affect your grade in Math 1LS3, nor will it affect how you are treated in any future math course(s) that you take at McMaster. Let us emphasize that the course instructors (Miroslav Lovric, Johannes Hofscheier, and Erin Clements) will not know the names of students who participate, or who do not participate in this research.

4. Possible Risks and Harms

There are no known or anticipated risks or discomforts associated with participating in this study.

If you feel uncomfortable about us using your responses for research, you have an opportunity to withdraw from this research, with no consequences. Thus, if you initially agree to be part of the study, but then change your mind, you can withdraw at any time up until 31 January 2019 by sending an email to the research assistant Reihaneh Jamalifar [redacted] Your data will be destroyed, and never used in our research.

Data must be kept for a minimum of 7 years as per institutional regulations. After this period, we will confidentially destroy all data we have collected (the data will be destroyed in the same way as your private information, your exams, and all your work with your name and/or student ID number are destroyed on campus).

5. Possible Benefits

Future generations of students taking Math 1LS3 will benefit from our study. In conducting this study, we hope to learn more about how computational thinking can be effectively integrated into an applied undergraduate calculus course to improve students' experiences and conceptual understanding in tertiary mathematics. We believe that this study will be valuable to the Department of Mathematics and Statistics at McMaster (and beyond) and could help inform teaching and pedagogical decisions, as well as stimulate possible curriculum changes.

Your participation could provide statistical evidence of improvement in your learning experiences and conceptual understanding, which might convince other instructors (possibly your future math or stats instructors at McMaster) to adopt similar techniques.

Furthermore, this study could be of use to math departments at other institutions with similar curriculum structure.

6. Voluntary Participation

Your participation in this study is completely voluntary. If you decide not to be part of the study, the research assistant will not include your responses in the data provided to the researchers. If you initially agree to be part of the study, but then change your mind, you can withdraw at any time up until 31 January 2019 by sending an email to the research assistant Reihaneh Jamalifar [redacted] If you decide to withdraw, there will be absolutely no consequences to you. In case you withdraw from the study, any data you have provided will be destroyed.

Let us emphasize that your participation, or withdrawal from participation, will not, in any way, affect your grade in Math 1LS3, nor will it affect how you are treated in any future math course(s) that you take at McMaster.

7. Confidentiality

You are participating in this study confidentially. Each student will be assigned a unique ID number code that will be used on their survey responses in place of a name. In an encrypted file, stored separately from the collected data, the link between student and participant ID is recorded, so that we can tie your consent to your work. But only the research assistant (and not any course instructors) will have access to this file. Raw data will be kept in an encrypted folder on a password-protected, personal laptop as to protect your privacy with a single back-up copy on an encrypted hard drive. Data must be kept for a minimum of 7 years as per institutional regulations. After this period has elapsed, all raw data will be destroyed (deleted).

Please note that representatives of The University of Western Ontario Non-Medical Research Ethics Board may require access to your study related records to monitor the conduct of the research.

8. Contacts for Further Information

If you require any further information regarding this research project or your participation in the study you may contact the principal investigator, Dr. George Gadanidis, [redacted]

If you have any questions about your rights as a research participant or the conduct of this study, you may contact The Office of Human Research Ethics [redacted]

This study has been reviewed by the McMaster University Research Ethics Board and received ethics clearance. If you have concerns or questions about your rights as a participant or about the way the study is conducted, please contact:

McMaster Research Ethics Secretariat
[redacted]
C/o Research Office for Administrative Development and Support
[redacted]

Appendix B: Developing Computer Coding Activities for Math 1LS3

When creating the computer labs for Math 1LS3, my initial challenge was deciding what concepts could be effectively remediated using a computational approach. While it would be relatively easy to incorporate computer technology in some capacity into many mathematical explorations, it was important for us to choose activities that provided unique, potentially transformative learning opportunities. For example, a “trivial” activity, though not without value, would be to ask students to create a program that would apply transformations to a standard function. This would be beneficial in reviewing the rules for transformations—and graphing functions in general—as well as providing students with more experience integrating coding and mathematics. However, we decided against including this activity since it involved a significant investment in preparation and students’ time, and it did not offer as many unique experiences as other activities. To entice students to invest significant amounts of their time, effort, and attention into the coding labs, we wanted to design activities with significant, far-reaching benefits, that is, those they could not experience when using a prepackaged application, such as Desmos or Maple.

In designing the labs, we specifically chose course material that students have historically struggled with conceptually, and not just technically. Once we had generated a list of topics, we considered how these concepts would be represented numerically and algorithmically. Some concepts, such as iterative processes used to solve equations or to find solutions of differential equations, lend themselves easily to reformulation for computation. However, certain topics (such as estimating solutions of an equation using the Intermediate Value Theorem) require more thought and preparation.

Next, we reformulated, when necessary, our models, techniques, and theorems from continuous versions to their discrete analogues (e.g., UV index, discussed briefly by Clements and Lovric (2018), in the section Discrete vs. Continuous Functions) and considered “provocative” questions we could ask to encourage students to reflect on, and deeply explore these concepts. We also considered the limitations that a shift to a computational model would bring, and anticipated the bridge students would have to make on their own to connect this representation to more abstract, theoretical ideas.

Example: Integration, Area, and Riemann Sums

In covering integral calculus in university courses, a significant amount of time is spent on techniques of integration, that is, on algebraic methods of evaluating definite and indefinite integrals.

Recall that a definite integral is defined as the limit of a Riemann sum:

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\Delta x, \quad \Delta x = \frac{b-a}{n}$$

where x_i^* is any sample point in the sub interval $[x_{i-1}, x_i]$.

Solving a definite integral using the Fundamental Theorem of Calculus requires that an algebraic formula for the antiderivative of $f(x)$ exists and that students are adequately skilled in integration techniques, which vary widely in their complexity and effectiveness. Alternatively, working with the summation notation on the right hand side of the definition to find an algebraic form for the Riemann sum and then evaluating the limit of this sum as n approaches infinity is generally a complex, if not impossible, task for first-year students.

Note that we can approximate the value of a definite integral using a finite Riemann sum:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i^*)\Delta x, \quad \Delta x = \frac{b-a}{n}$$

where x_i^* is any sample point in the sub interval $[x_{i-1}, x_i]$.

(Recall that a definite integral can be interpreted as the net or signed area of the region bounded by the graph of the function and the horizontal axis over a finite interval. The area of this bounded region can be estimated by computing the sum of the areas of approximating rectangles, that is, a Riemann sum.)

Even though evaluating a finite sum is significantly less complicated (conceptually, and otherwise) than evaluating an infinite sum, students nevertheless struggle with this concept. Faced with having to calculate a finite sum algebraically, students attempt to memorize and use abstract formulas found in their textbook and ultimately make conceptual, in addition to technical, errors. Their challenges are further exacerbated by their lack of familiarity with the summation notation, in particular with the role and purpose of the index of summation.

One error we have witnessed many times is students using all endpoints in both sums, thus instead of using n rectangles as required, they use $n + 1$ rectangles, with the extra rectangle being formed outside of the bounded region. The source of this error could lie in the fact that, although this is a geometric situation, students rely on algebraic reasoning based on their (mis)interpretation of the formulas for the left and the right sums. Without drawing the region and corresponding rectangles, students would not recognize the nature of their error, and might assume it was just a "small" calculation error. For smaller values of n, we often ask students to draw the bounded region and corresponding approximating rectangles, thus visualizing the sum as well as representing it algebraically. However, as the graphs of functions become more complicated, and as the number of rectangles becomes large, representing the Riemann sum geometrically using pen and paper becomes time-consuming and is often omitted altogether.

From a technical standpoint, reformulating a definite integral for computation is straightforward since we already have a discrete representation for an approximation—that is, a finite Riemann sum—established. With the technical aspects supported by computer technology, students are free to explore more theoretical questions about Riemann sums and their relation to integration and definite integrals, which enhances their understanding of this complex object. For instance, the following questions could stimulate their explorations and reflection:

- Why does the endpoint at which we choose to calculate the height of an approximating rectangle on a given subinterval become insignificant as the number of approximating rectangles approaches infinity?

- For a given continuous function on a finite integral, which of L_{10}, L_5, R_5, R_{10} would produce the largest value? Why?
- Would the finite sum using the height calculated from the midpoint of each subinterval equal the average of the left and right sums on the same interval, using the same number of rectangles? Why or why not?
- How could we find the area of the bounded region between two curves on a given interval?
- Suppose the region bounded by the curve $y = f(x)$, $y = 0$, $x = a$, and $x = b$ was rotated around the x-axis to form a solid. How could you approximate the volume of this solid using a Riemann sum?

As well, since students have historically struggled with this topic, we felt that multiple complementary representations of this concept could help support their understanding and potentially offer transformative, rewarding learning experiences. With our ideas in mind, we felt integration and related concepts and applications such as Riemann sums or area between curves was an ideal topic to be remediated with computation.

Appendix C contains the full set of Lab 3 coding activities developed for Math 1LS3 students in the fall 2018 semester. This lab was designed to complement our study of initial value problems (Euler’s method) and definite integrals (Riemann sums).

Appendix C: Computer Lab 3

Lab 3 (IVPs and Riemann Sums)

There are seven questions, worth 1 point each.

Enter your answers into the """ online form/assessment tool (https://www.childsmath.ca/childsa/forms/main_login.php)""" (log in with your Mac ID and password).

You will need to login before you start working on Problem 1.

There is no submit button. Save your work often. You can change your answers as long as the lab is open (so, you can start your lab today, do part of it, and save; then come back tomorrow, fix the answer that you realized was incorrect, and continue working on the lab). The answers that are there at the moment when the lab closes are taken as your final answers. </b>

```
In [ ]: # REMEMBER TO RUN THIS CELL EVERY TIME YOU COME BACK TO THIS NOTEBOOK

        # import packages and assign shortcuts that we will be using in 1LS3
        import matplotlib.pyplot as plt # MatPlotLib: Plotting library for Python
        import numpy as np # NumPy: N-dimensional array for numerical computation
        %matplotlib inline
        import pandas as pd  # to deal with data

        # TO START, RUN THIS CELL

        print(" ")
        print("LAB 3!!!!")
        print(" ")
```

Euler's Method

Euler's Method is used to generate approximate values of the solution to an initial value problem (IVP) of the form

$\frac{dy}{dt} = G(t, y)$, where $y(t_0) = y_0$.

Algorithm:

t_0 and y_0 are given by the initial condition. To compute successive approximations, use:

$t_{n+1} = t_n + \Delta t$

$y_{n+1} = y_n + G(t_n, y_n)\Delta t$

where $n = 0, 1, 2, 3,$ and so on, and Δt is the step size.

Note: this is how the formula is often given in literature. In the code that we use, we will use the following variation:

$t_i = t_{i-1} + \Delta t$

$y_i = y_{i-1} + G(t_{i-1}, y_{i-1})\Delta t$

where $i = 1, 2, 3, 4,$ and so on. Make sure to understand that the two formulas are identical.

Example: Applying Euler's Method to a Linear Differential Equation

Consider the initial value problem $\frac{dy}{dt} = t + y$, where $y(0) = 1$.

Our task is to use Euler's Method to approximate the value $y(2)$ and compare it with the true value (given by a formula).

(a) Examine the code below and run it to see what it does. In particular, identify the step size.

(b) Show that $y(t) = 2e^t - t - 1$ is the solution of the given IVP. In this case the differential equation is simple enough and we can solve it algebraically. As a benefit, we will be able to compare Euler's Method approximations of the solution with the true solution.

(c) Modify the code below to estimate the value of $y(2)$ using a step size of 0.25. How many steps will you need to compute?

(d) Modify the code below so that is uses 20 steps. What is the approximate value of $y(2)$? Repeat with 100 steps. What is the approximate value of $y(2)$? Repeat with 1000 steps. What is the approximate value of $y(2)$?

In [ ]:
```
# Euler's method

# define variables and parameters
t = []
y = []
n = 100 # number of steps
deltat = 0.05  # step size

# define initial values and append to arrays
t0 = 0
y0 = 1
t.append(t0)
y.append(y0)

# Euler's method algorithm
for i in range(1,n+1):
    ti = t[i-1] + deltat
    yi = y[i-1] + (t[i-1] + y[i-1])*deltat # make sure you understand this line!
    t.append(ti)
    y.append(yi)

# true solution (in Math 1LT3 we learn how to calculate it)
algebraic=2*np.exp(t)-t-1

# if desired: print the table of values for y
# print(y)

# output statement
print("Euler: When t is", t[n], "the y-value is approximately", y[n])
print("Algebraic (true) solution: When t is", t[n], "the y-value is", algebraic
[n])
print("The difference between algebraic (true) and Euler (approximation) is", np.a
bs(algebraic[n]-y[n]))

# plot solution
plt.plot(t,y,label='Euler',color='b')
plt.plot(t,algebraic,label='algebraic',color='g')
plt.scatter(t[n], y[n], color='b')
plt.scatter(t[n], algebraic[n], color='g')
plt.title("Approximate Solution to an IVP using Euler's Method")
plt.xlabel("t")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.show()
```

Problem 1 (Applying Euler's Method to Caribou Population Dynamics)

Consider the initial value problem

$$P'(t) = 0.24P(t)\left(1 - \frac{P(t)}{720}\right)\left(\frac{P(t)}{216} - 1\right)$$

where $P(t)$ represents the number of caribou in an ecosystem around Kuujjuaq in Northern Quebec, and the time t is in years.

See Problem 1 in the online form/assessment tool (i.e., where you enter your answers) for the value of A that you need to answer the question.

The number A is your initial condition, i.e., $P(0) = A$. Adjust your code from the previous example to estimate the value of $P(50)$. What does the model say about the population over time, i.e., as $t \to \infty$?

In [ ]:
```
# Modelling caribou population using modified logistic differential equation

### A = 338 <- i got this from your screenshot

# define variables and parameters
t = []
P = []
n = 10000 # number of steps
deltat = 0.005 # step size

# define initial values and append to arrays
t0 = 0
P0 = 338   # replace A by the number you are given
t.append(t0)
P.append(P0)

# Euler's method algorithm
for i in range(1,n+1):
    ti = t[i-1] + deltat
    Pi = P[i-1] + (0.24*P[i-1]*(1-P[i-1]/720)*(P[i-1]/216-1))*deltat # I modified
this line to reflect our new P'
    t.append(ti)
    P.append(Pi)

# output statement
print("The population after", t[n], "years is approximately",  P[n], "individual
s.")

# plot solution
plt.plot(t,P)
plt.scatter(t[n], P[n], color='r')
plt.title("Approximate Solution To a Modified Logistic Model")
plt.xlabel("time")
plt.ylabel("population")
plt.grid()
plt.show()
```

Problem 2 (Modelling TBC outbreak)

In the paper *Bi-logistic model for disease dynamics caused by Mycobacterium tuberculosis in Russia* the authors A. I. Lavrova, E. B. Postnikov, O. A. Manicheva, and B. I. Vishnevsky need to solve the differential equation [slightly modified here]

$$\frac{dR(t)}{dt} = \frac{1}{\tau}\left(1 - 1.15R(t) - 0.9e^{-k\tau\, R(t)}\right)$$

for the cumulative ratio of people suffering from an outbreak of Mycobacterium tuberculosis (TBC for short). The time t is measured in days.

Initially, no one is infected, so $R(0) = 0$. The values of the parameters are $\tau = 1.23$ and $k = 2.09$.

Run Euler's Method with an appropriate number of steps to obtain an approximation for $R(10)$ correct to three decimal places. (Recall that "correct to three decimal places" does not mean to round off the third decimal place but instead to quote the first three correct decimals; for example, if Python returns 0.238776125, then your answer should be 0.238 and not 0.239).

How do you know what's an appropriate number of steps? Run the code with an increasing number of steps (of course, each time adjusting the step size) until the approximation stabilizes, i.e., the first three decimal places do not change.

In [ ]:
```
# Applying Euler's Method to an Autonomous DE (Logistic Model)

# define variables and parameters
t = []
R = []
n = 10000 # number of steps
deltat = 10/n # step size

tau=1.23
k=2.09

# define initial values and append to arrays
t0 = 0
R0 = 0
t.append(t0)
R.append(R0)

# Euler's method algorithm
for i in range(1,n+1):
    ti = t[i-1] + deltat
    Ri = R[i-1] + ((1/tau)*(1-1.15*R[i-1]-0.9*np.exp(-k*tau*R[i-1])))*deltat # mak
e sure you understand this line!
    t.append(ti)
    R.append(Ri)

# output statement
print("The ratio after", t[n], "years is approximately",  R[n])
```

Problems 3,4 (Modelling acetaminophen overdose)

The paper *Mathematical modeling of liver injury and dysfunction after acetaminophen overdose: Early discrimination between survival and death* models liver damage caused by the acetaminophen overdose (see slides and paper posted on our course webpage). The first in the system of differential equations studied

$$\frac{dA(t)}{dt} = -\frac{\alpha}{H_{\text{max}}} A(t)H(t) - \delta_a A(t)$$

models the amount $A(t)$ of acetaminophen over time (measured in hours). Experimental results produced the formula $H(t) = 0.17(\sin 3.7t)^2$ for the number of functional hepatocytes; these hepatocytes contribute to the decrease in $A(t)$.

With this formula for $H(t)$, the differential equation for $A(t)$ becomes

$$\frac{dA(t)}{dt} = -\frac{\alpha}{H_{\text{max}}} \cdot 0.17 \cdot (\sin 3.7t)^2 \cdot A(t) - \delta_a A(t)$$

Assume that $A(0) = 10^6$. The parameter values are $\alpha = 6.3$, $H_{\text{max}} = 1.6$, and $\delta_a = 0.33$.

(**Problem 3**) Use Euler's method with the step size $\Delta t = 0.002$ to find an approximation of $A(4)$. Round off to the nearest integer.

(**Problem 4**) Plot the graph of $A(t)$ for $0 \le t \le 4$. How can you describe the shape of the graph?

In [ ]:
```
# acetaminophen overdose model

# define variables and parameters
t = []
A = []
n = 2000 # number of steps
deltat = 0.002 # step size

alpha = 6.3
Hmax = 1.6
deltaa = 0.33

# define initial values and append to arrays
t0 = 0
A0 = 10**6
t.append(t0)
A.append(A0)

# Euler's method algorithm
for i in range(1,n+1):
    ti = t[i-1] + deltat
    Ai = A[i-1] + ((-alpha/Hmax)*0.17*(np.sin(3.7*t[i-1]))**2*A[i-1]-deltaa*A[i-
1])*deltat
    t.append(ti)
    A.append(Ai)

# output statement
print("The value after", t[n], " is approximately",  A[n])

# plot solution
plt.plot(t,A)
plt.scatter(t[n], A[n], color='r')
plt.title("Approximate Solution")
plt.xlabel("time")
plt.ylabel("acetaminophen")
plt.grid()
plt.show()
```

Riemann sums

First, we work on a couple of pieces of useful code.

To start, we ask Python to sketch the graph of $y = e^{-x^2}$ on $[-3, 3]$.

In [ ]:
```
# study this code to review how to plot the graph of a function
# on a given interval (remember that a function is defined as a table of values)

# define variables and parameters
a = -3
b = 3
n = 240
x = np.linspace(a,b,n)

# define function
y = np.exp(-x**2)

# print(x) # to see the values of x
# print(y) # to see the values of the function y

# plot function
plt.plot(x,y, color = 'navy', label="$y=e^{-x^2}$")
plt.xlabel('x')
plt.ylabel('y')
plt.title("The Bell Curve")
plt.legend()
plt.grid()
plt.show()
```

Adding numbers

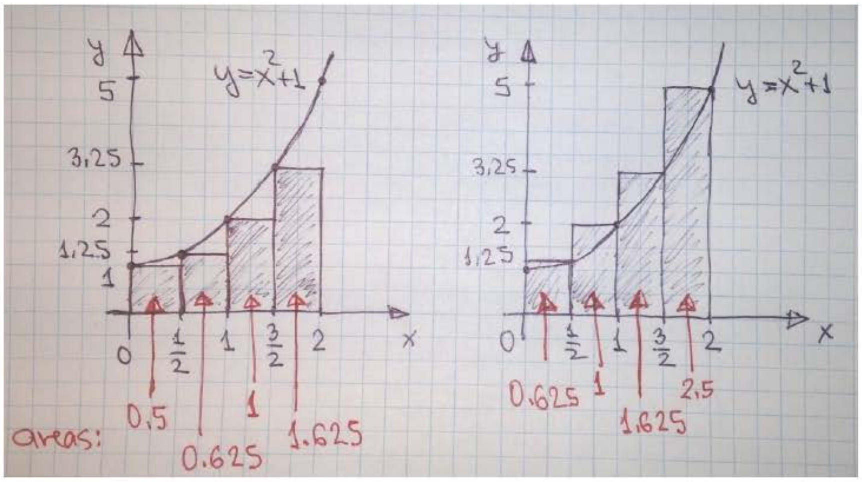
Python has a cool way of adding numbers in a list - very useful!

In [ ]:
```
# go through the code in this cell, and look at the output

w=[2,3,4,5,6,7,8]  # define a list, call it w
print(w)  # recall that we start from 0; thus w[0]=2, w[2]=4, and so on

a=sum(w[0:3]) # keep in mind that the range [m:n] starts with m, but the last valu
e is n-1
print(a)  # so a = w[0]+w[1]+w[2] = 2+3+4

a=sum(w[5:6]) # there is only one term in this sum, w[5]
print(a)

a=sum(w[2:6]) # a = w[2]+w[3]+w[4]+w[5] = 4+5+6+7
print(a)

b=np.linspace(1,100,100)  # the list b has 100 elements: b[0]=1, b[1]=2, ... , b[9
9]=100
print(b)
a=sum(b[0:100]) # computes the sum b[0]+b[1]+...+b[99]
print(a)
```

Computing Riemann sums

Next, we review how to compute Riemann sums. We start with a simple example.

On a piece of paper, sketch the function $y = x^2 + 1$ on $[0, 2]$, and draw the rectangles that belong to L_4 and R_4 (left and right Riemann sums with four rectangles).



Computing Riemann sums - version 1

Keeping in mind the two pictures above, study the code in the cell below which computes L_4 and R_4 for $y = x^2 + 1$ on $[0, 2]$.

In [ ]:

```
# Riemann sums

# define variables and parameters
a = 0
b = 2 # interval [0,2]
n = 4 # number of approximating rectangles

x = np.linspace(a,b,n+1)
print("x values are:",x)  #run the code now and look at the pictures above.
          # .. so these are the x values that divide the given interval into 4 sub
intervals

# define the function y=f(x), i.e., the 'heights' of approximating rectangles
y=x**2+1
print("the corresponding values of the function y=x^2+1 are:",y)

# compute the base length of each rectangle
delta_x = (b-a)/n
print("the base length of each rectangle is",delta_x)

# compute areas of approximating rectangles
# recall: the area of the approximating rectangle is base length * height = delta_
x * the value of the function

areas = y*delta_x  # this command computes all areas at once, and stores them in t
he list called areas
print("the areas of the approximating rectanges are",areas)

# look at the pictures above, where the areas are recorded in red

# compute the left-sum (think about which areas we have to add - look at the pictu
re again)
ls = sum(areas[0:n])
print("The left sum is:",ls)

# compute right-sum
rs = sum(areas[1:n+1])
print("The right sum is:",rs)
```

Problem 5

Modify the code above to approximate $\int_{-1}^{2} e^{-x^2}\,dx$ using R_8 and L_8. You will see that R_8 and L_8 are not close to each other.

Keep increasing the number of rectangles until you can be sure that you have approximated the given integral correctly to four decimal places. What is the value of the integral?

In [ ]:

```
# Riemann sums - basic code

a = -1
b = 2
n = 800000
delta_x = (b-a)/n

x = np.linspace(a,b,n+1)
y = np.exp(-x**2)

areas = y*delta_x

ls = sum(areas[0:n])
print("The left sum is:",ls)

rs = sum(areas[1:n+1])
print("The right sum is:",rs)
```

Computing Riemann sums - version 2

Study the code in the cell below which computes L_4 and R_4 for $y = x^2 + 1$ on $[0, 2]$, using arrays and loops.

Modify the code to find M_4. How does it relate to L_4 and R_4?

In [ ]:
```
# Riemann sums, code using arrays and loops

# define variables and parameters
a = 0
b = 2
n = 4
x = np.linspace(a,b,n+1)

# define step size
delta_x = (b-a)/n

# define arrays for y-values     ## create an array for the y-values at the midpoi
nts
y_left = []
y_right = []
y_mid = []

# left sum
for i in range(n):
    xl = x[i]
    yl = xl**2+1
    y_left.append(yl)
ls = sum(y_left)*delta_x
print("The left sum is:", ls)

# right sum
for i in range(n):
    xr = x[i+1]
    yr = xr**2+1
    y_right.append(yr)
rs = sum(y_right)*delta_x
print("The right sum is:", rs)

# midpoint sum
for i in range(n):
    xm = (x[i]+x[i+1])/2
    ym = xm**2+1
    y_mid.append(ym)
ms = sum(y_mid)*delta_x
print("The midpoint sum is:", ms)
```

Problems 6, 7

Modify the code above to approximate $\int_{-1}^{2} e^{-x^2}\,dx$ using M_8.

(**Problem 6**) What is the value of M_8? Give your answer correct to four decimal places.

(**Problem 7**) Does M_8 equal the average of L_8 and R_8?

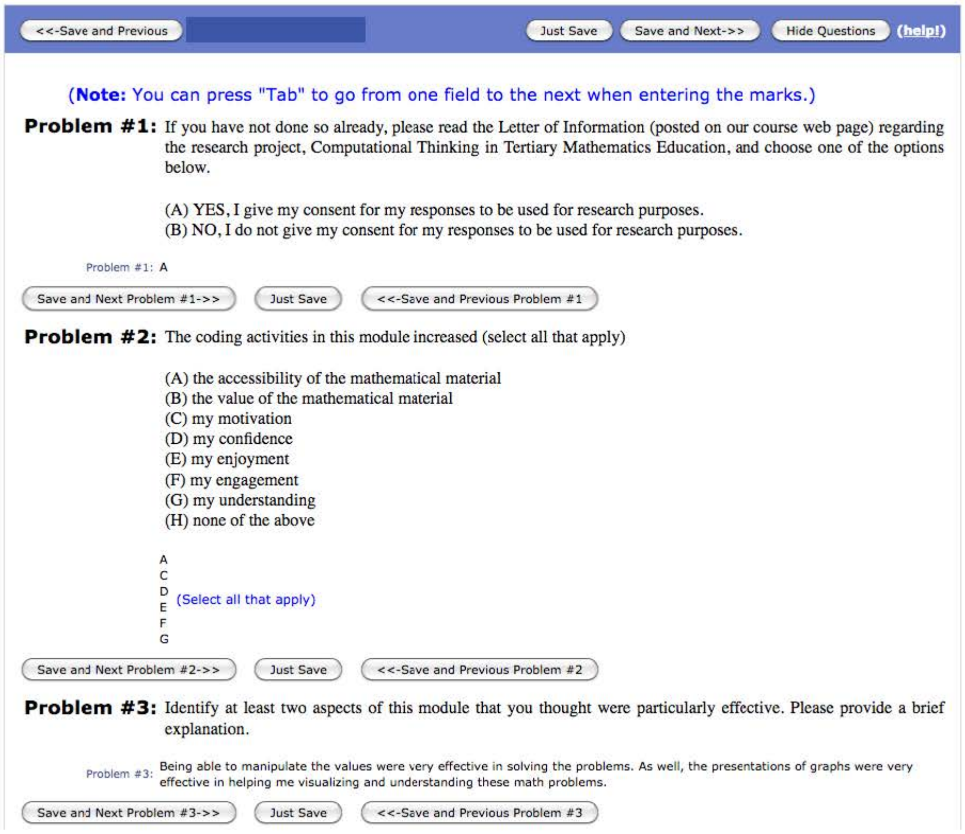
In [ ]:

```
# Riemann sums, code using arrays and loops

# define variables and parameters
a = -1
b = 2
n = 8
x = np.linspace(a,b,n+1)

# define step size
delta_x = (b-a)/n

# define arrays for y-values     ## create an array for the y-values at the midpoi
nts
y_left = []
y_right = []
y_mid = []

# left sum
for i in range(n):
    xl = x[i]
    yl = np.exp(-xl**2)
    y_left.append(yl)
ls = sum(y_left)*delta_x
print("The left sum is:", ls)

# right sum
for i in range(n):
    xr = x[i+1]
    yr = np.exp(-xr**2)
    y_right.append(yr)
rs = sum(y_right)*delta_x
print("The right sum is:", rs)

# midpoint sum
for i in range(n):
    xm = (x[i]+x[i+1])/2
    ym = np.exp(-xm**2)
    y_mid.append(ym)
ms = sum(y_mid)*delta_x
print("The midpoint sum is:", ms)
```

End of Lab 3

Appendix D: Survey Questions

Appendix D 1: Sample Responses to Lab 1 Survey Questions


<<-Save and Previous
Just Save
Save and Next->>
Hide Questions
(help!)
Save and Next Problem #1->>
Just Save
<<-Save and Previous Problem #1
Save and Next Problem #2->>
Just Save
<<-Save and Previous Problem #2
Save and Next Problem #3->>
Just Save
<<-Save and Previous Problem #3

[redacted]

(**Note:** You can press "Tab" to go from one field to the next when entering the marks.)

Problem #1: If you have not done so already, please read the Letter of Information (posted on our course web page) regarding the research project, Computational Thinking in Tertiary Mathematics Education, and choose one of the options below.

(A) YES, I give my consent for my responses to be used for research purposes.
(B) NO, I do not give my consent for my responses to be used for research purposes.

Problem #1: A

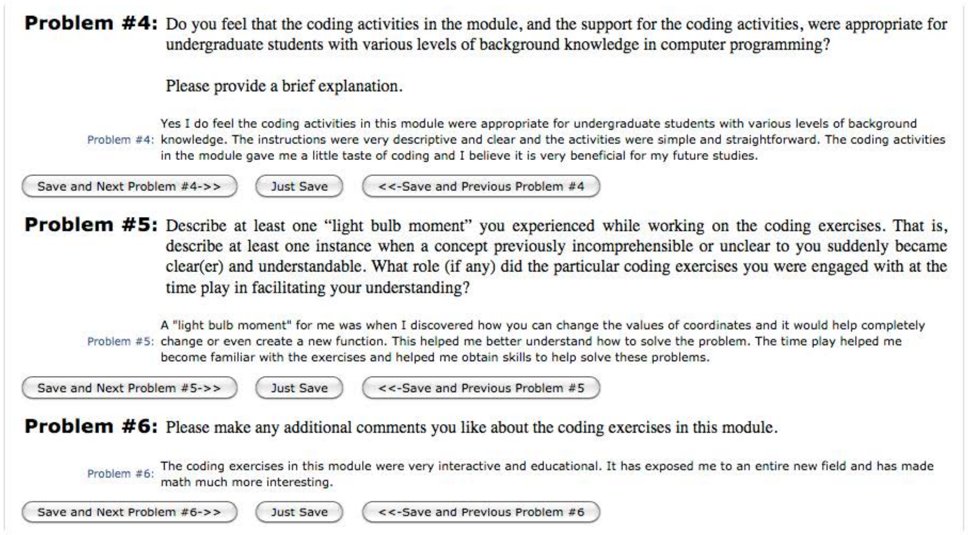
Problem #2: The coding activities in this module increased (select all that apply)

(A) the accessibility of the mathematical material
(B) the value of the mathematical material
(C) my motivation
(D) my confidence
(E) my enjoyment
(F) my engagement
(G) my understanding
(H) none of the above

A
C
D
E (Select all that apply)
F
G

Problem #3: Identify at least two aspects of this module that you thought were particularly effective. Please provide a brief explanation.

Problem #3: Being able to manipulate the values were very effective in solving the problems. As well, the presentations of graphs were very effective in helping me visualizing and understanding these math problems.



Problem #4: Do you feel that the coding activities in the module, and the support for the coding activities, were appropriate for undergraduate students with various levels of background knowledge in computer programming?

Please provide a brief explanation.

Problem #4: Yes I do feel the coding activities in this module were appropriate for undergraduate students with various levels of background knowledge. The instructions were very descriptive and clear and the activities were simple and straightforward. The coding activities in the module gave me a little taste of coding and I believe it is very beneficial for my future studies.

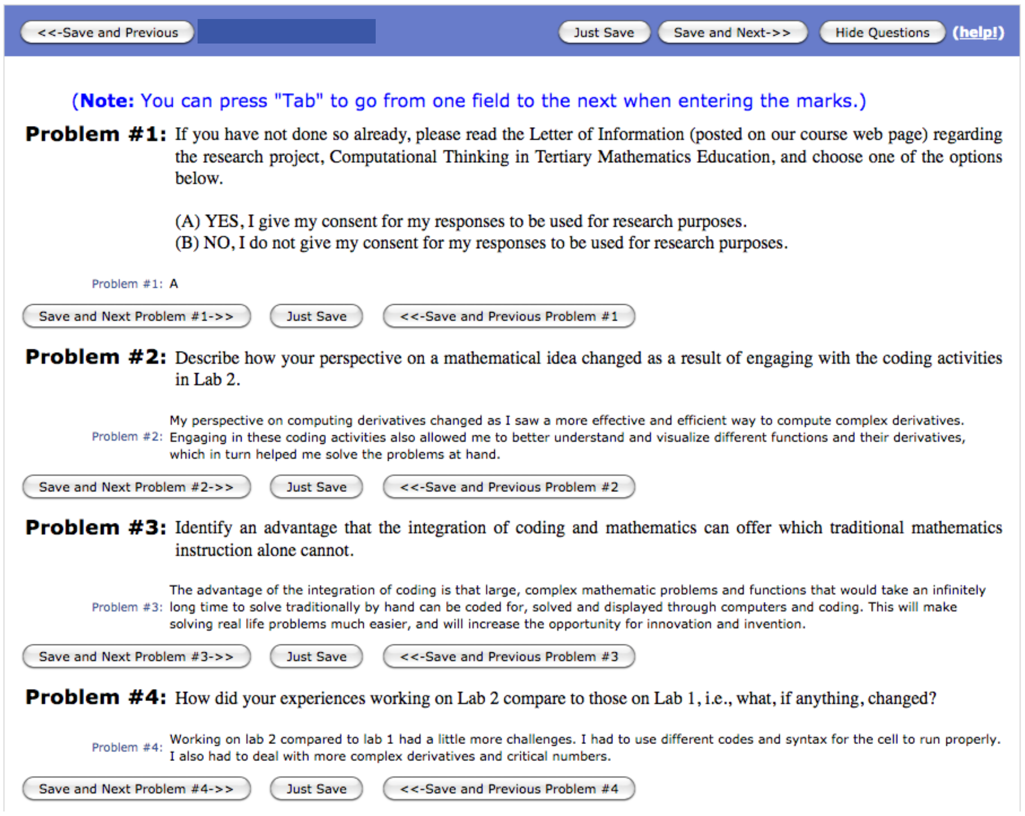
Problem #5: Describe at least one "light bulb moment" you experienced while working on the coding exercises. That is, describe at least one instance when a concept previously incomprehensible or unclear to you suddenly became clear(er) and understandable. What role (if any) did the particular coding exercises you were engaged with at the time play in facilitating your understanding?

Problem #5: A "light bulb moment" for me was when I discovered how you can change the values of coordinates and it would help completely change or even create a new function. This helped me better understand how to solve the problem. The time play helped me become familiar with the exercises and helped me obtain skills to help solve these problems.

Problem #6: Please make any additional comments you like about the coding exercises in this module.

Problem #6: The coding exercises in this module were very interactive and educational. It has exposed me to an entire new field and has made math much more interesting.

Appendix D 2: Sample Responses to Lab 2 Survey Questions



(**Note:** You can press "Tab" to go from one field to the next when entering the marks.)

Problem #1: If you have not done so already, please read the Letter of Information (posted on our course web page) regarding the research project, Computational Thinking in Tertiary Mathematics Education, and choose one of the options below.

(A) YES, I give my consent for my responses to be used for research purposes.
(B) NO, I do not give my consent for my responses to be used for research purposes.

Problem #1: A

Problem #2: Describe how your perspective on a mathematical idea changed as a result of engaging with the coding activities in Lab 2.

Problem #2: My perspective on computing derivatives changed as I saw a more effective and efficient way to compute complex derivatives. Engaging in these coding activities also allowed me to better understand and visualize different functions and their derivatives, which in turn helped me solve the problems at hand.

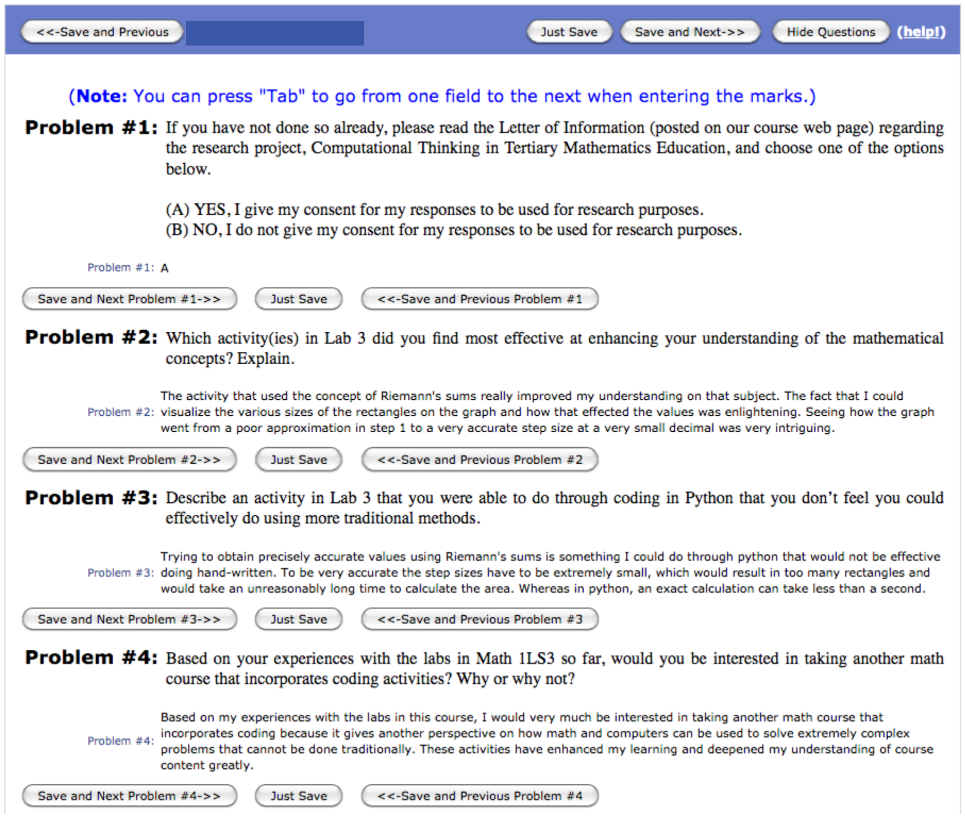
Problem #3: Identify an advantage that the integration of coding and mathematics can offer which traditional mathematics instruction alone cannot.

Problem #3: The advantage of the integration of coding is that large, complex mathematic problems and functions that would take an infinitely long time to solve traditionally by hand can be coded for, solved and displayed through computers and coding. This will make solving real life problems much easier, and will increase the opportunity for innovation and invention.

Problem #4: How did your experiences working on Lab 2 compare to those on Lab 1, i.e., what, if anything, changed?

Problem #4: Working on lab 2 compared to lab 1 had a little more challenges. I had to use different codes and syntax for the cell to run properly. I also had to deal with more complex derivatives and critical numbers.

Appendix D 3: Sample Responses to Lab 3 Survey Questions



(**Note:** You can press "Tab" to go from one field to the next when entering the marks.)

Problem #1: If you have not done so already, please read the Letter of Information (posted on our course web page) regarding the research project, Computational Thinking in Tertiary Mathematics Education, and choose one of the options below.

(A) YES, I give my consent for my responses to be used for research purposes.
(B) NO, I do not give my consent for my responses to be used for research purposes.

Problem #1: A

Problem #2: Which activity(ies) in Lab 3 did you find most effective at enhancing your understanding of the mathematical concepts? Explain.

Problem #2: The activity that used the concept of Riemann's sums really improved my understanding on that subject. The fact that I could visualize the various sizes of the rectangles on the graph and how that effected the values was enlightening. Seeing how the graph went from a poor approximation in step 1 to a very accurate step size at a very small decimal was very intriguing.

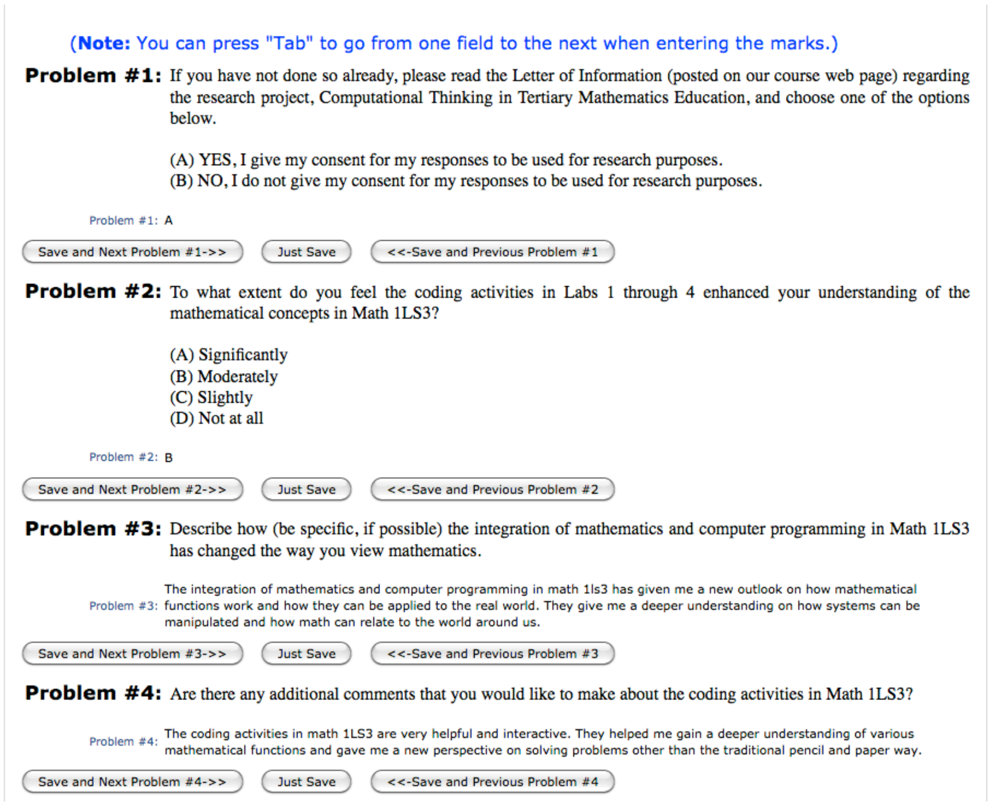
Problem #3: Describe an activity in Lab 3 that you were able to do through coding in Python that you don't feel you could effectively do using more traditional methods.

Problem #3: Trying to obtain precisely accurate values using Riemann's sums is something I could do through python that would not be effective doing hand-written. To be very accurate the step sizes have to be extremely small, which would result in too many rectangles and would take an unreasonably long time to calculate the area. Whereas in python, an exact calculation can take less than a second.

Problem #4: Based on your experiences with the labs in Math 1LS3 so far, would you be interested in taking another math course that incorporates coding activities? Why or why not?

Problem #4: Based on my experiences with the labs in this course, I would very much be interested in taking another math course that incorporates coding because it gives another perspective on how math and computers can be used to solve extremely complex problems that cannot be done traditionally. These activities have enhanced my learning and deepened my understanding of course content greatly.

Appendix D 4: Sample Responses to Lab 4 Survey Questions



(**Note:** You can press "Tab" to go from one field to the next when entering the marks.)

Problem #1: If you have not done so already, please read the Letter of Information (posted on our course web page) regarding the research project, Computational Thinking in Tertiary Mathematics Education, and choose one of the options below.

(A) YES, I give my consent for my responses to be used for research purposes.
(B) NO, I do not give my consent for my responses to be used for research purposes.

Problem #1: A

Problem #2: To what extent do you feel the coding activities in Labs 1 through 4 enhanced your understanding of the mathematical concepts in Math 1LS3?

(A) Significantly
(B) Moderately
(C) Slightly
(D) Not at all

Problem #2: B

Problem #3: Describe how (be specific, if possible) the integration of mathematics and computer programming in Math 1LS3 has changed the way you view mathematics.

Problem #3: The integration of mathematics and computer programming in math 1ls3 has given me a new outlook on how mathematical functions work and how they can be applied to the real world. They give me a deeper understanding on how systems can be manipulated and how math can relate to the world around us.

Problem #4: Are there any additional comments that you would like to make about the coding activities in Math 1LS3?

Problem #4: The coding activities in math 1LS3 are very helpful and interactive. They helped me gain a deeper understanding of various mathematical functions and gave me a new perspective on solving problems other than the traditional pencil and paper way.

Curriculum Vitae

Name: Erin Clements

Post-secondary Education and Degrees:

McMaster University
Hamilton, Ontario, Canada
1999-2004 B.Sc.

McMaster University
Hamilton, Ontario, Canada
2008-2010 M.Sc.

Western University
London, Ontario, Canada
2016-2020 Ph.D.

Related Work Experience

Lecturer
Department of Mathematics and Statistics
McMaster University
2006-present

Research Assistant
Faculty of Education
Western University
2016-2020

Mathematics Teacher
Columbia International College
2004-2012

Publications:

Clements, E. & Lovric, M. (2018). Chapter 3: Interdisciplinary Approaches Motivate and Enrich Teaching and Learning Mathematics. In Sibbald, T. (Ed.) *Teaching Interdisciplinary Mathematics (Learning to Teach, Teaching to Learn).* University of Illinois: Common Ground Publishing.

Gadanidis, G., Clements, E., & Yiu, C. (2018). Group Theory, Computational Thinking, and Young Mathematicians. *(The Journal of) Mathematical Thinking and Learning, 20*(1), 32-53. doi: 10.1080/10986065.2018.1403542