

UiO : **University of Oslo**

Odd Petter Sand

# **Integrating Computing with Mathematics and Science Education**

Case Studies of Student Understanding and  
Teaching Design

**Thesis submitted for the degree of Philosophiae Doctor**

Centre for Computing in Science Education (CCSE)  
Faculty of Mathematics and Natural Sciences



**2021**

© **Odd Petter Sand, 2021**

*Series of dissertations submitted to the  
Faculty of Mathematics and Natural Sciences, University of Oslo  
No. 2448*

ISSN 1501-7710

All rights reserved. No part of this publication may be reproduced or transmitted, in any form or by any means, without permission.

Cover: Hanne Baadsgaard Utigard.  
Print production: Representralen, University of Oslo.

*I dedicate this thesis, all my work, to my girlfriend, Linn, who will have to support me and our future children, dog, and cat once it<sup>1</sup> gets released into the public.*

---

<sup>1</sup>The work, not the cat.



# Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* at the University of Oslo. The research presented here was conducted at the University of Oslo, under the supervision of professor Knut Mørken, associate professor Marcos D. Caballero, associate professor Elise Lockwood and lecturer Christine Lindstrøm. This work was supported by the Norwegian Agency for International Cooperation and Quality Enhancement in Higher Education (DIKU) through the Centres for Excellence in Education (SFU) program (grant no. SFU-2016/10004).

The thesis is a collection of three papers, presented in chronological order of writing. The common theme to them is the integration of computing (computer programming) into mathematics and science education. These exploratory case studies focus on students' understanding and the design of teaching experiments. The papers are accompanied by an extended abstract ("kappe") that relates them to each other and provides background information and motivation for the work. All the papers are written together with my supervisors. Additionally, Tor Ole B. Odden is a co-author of the first paper.

## Acknowledgements

The work you are about to read would not have been possible without the contributions of my supervisors:

*Knut:* You somehow always found time for questions and discussions, even with your crazy schedule as Vice Dean of Education at the Faculty. You have a unique talent for seeing and helping people, and your wisdom and compassion makes you a role model for supervisors everywhere. I would recommend you to just about anyone looking for a supervisor if your schedule would allow it.

*Danny:* You have been part of the project since the beginning, and always contributed with excellent questions and observations, no matter which direction the work took. I deeply appreciate your support, your dedication to the quality of the work and your sense of humour.

*Christine:* During my first year at CCSE, you were a constant source of support and good advice that kept me going through difficult times. Your steadying presence was a very important contributing factor to seeing this work through to the end.

*Elise:* With your perspective on things and your inspiring research, I finally found the direction in my work that had been missing. Thank you for all your contributions to the writing, for introducing me to the world of math ed research, for all the online meetings where you had to get up at an ungodly hour because of time zones, and for the endless amusement your cats provided in those meetings.

In addition, heartfelt thanks to the following people:

*Professor Emeritus Joe Redish:* Your writing and teaching designs have been a massive inspiration for my work, and demonstrated what education research is all about. The inspiring dinner conversation at the AAPT/PERC 2018 Conference left me a little starstruck, but I am deeply grateful for you taking the time to discuss work in the field with the next generation of researchers.

*Director of the CCSE, Anders Malthe-Sørenssen:* Thank you for giving me the chance to prove myself as a researcher, and for generously extending my grant when the COVID-19 pandemic threw a spanner in the works.

*Head of Office at the CCSE, Tone Skramstad:* Thank you for keeping us organised, focused and for handling all the administrative things even when it must have felt like herding cats at times. And for all the waffles!

*Cathrine Tellefsen:* Thank you for all the interesting conversations about teaching and education, and for encouraging me to finish what I started. Your passion for enabling students to realise their full potential is something all educators should aspire to.

*Tor Ole Odden:* Thank you for getting me back on my feet and helping me finish my first paper. Your vision and dedication to excellence makes you a force of nature in the field of physics education research, and I very much enjoyed your insight and perspective on things in our conversations.

*Victoria Haynes:* Thank you for all our good talks about life and work and everything else. You bring a unique perspective to everything you do, and these four years would have been considerably less colourful without your indomitable cheerfulness and all the friendly banter over football.

---

*Lex Nederbragt:* It was a joy to work with you in teaching programming and math to life science students, and it meant a lot that you were so open to and appreciative of all my suggestions. I really admire the work you put into making mathematics and computing understandable to your students, and I hope this thesis can contribute to that in some way.

*Even, Simon, Simen, Eli, Øyvind, and the other friendly people I shared an office space with:* Thank you for all the good conversations, the laughs, the coffee breaks. When we all had to work from home, you were the ones that I missed the most. Special shout-out to *Simon* for introducing me to our D&D group and the world of Critical Role. Who knew that swindling nobles, stealing ships and putting on a play in a tavern could be so much fun?

*The LaTeX gurus at the University Library:* Your grasp of obscure, arcane knowledge got me through a maze of unhelpful error messages. For a moment there, you know, I was actually questioning the wisdom of auto-converting my entire thesis and all its references from Word and Zotero in the final week. Without you, I would not have had time to write these acknowledgements.

*Mum and Dad:* Thanks for providing my cells with the recipe to build a human being, for your endless support and patience, and for taking such good care of Puma. I'm sorry I lied about having done all my homework in sixth grade, but thank you for letting me keep the Nintendo console you got me anyway.

*Linn, love of my life:* Thank you for your support, for believing in me and for all the help with the text on the manuscripts. Your incredible eye for detail and your mad editing skillz bring out the best in my work, much as you bring out the best in me. I feel like the luckiest man in the vastness of space and time for getting to spend my days on this little planet with you.

**• Odd Petter Sand**

Oslo, September 2021





# List of Papers

## Paper I

Sand, O. P., Odden, T. O. B., Lindstrøm, C. & Caballero, M. D. “How Computation Can Facilitate Sensemaking About Physics: A Case Study”. In: *2018 Physics Education Research Conference Proceedings* (2018) <https://www.compadre.org/per/items/detail.cfm?ID=14850>

## Paper II

Sand, O. P., Lockwood, E., Caballero, M. D. & Mørken, K. “Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing”. *Submitted for publication. Updated preprint available at: <https://edarxiv.org/n8svc/>*

## Paper III

Sand, O. P., Lockwood, E., Caballero, M. D. & Mørken, K. “Students’ Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study”. *Submitted for publication. Updated preprint available at: <https://edarxiv.org/hrq78>*



# Contents

Preface	iii
List of Papers	vii
Contents	ix
List of Figures	xi
List of Tables	xiii
List of Code Samples	xv
<b>1 Introduction</b>	<b>1</b>
1.1 Why?	1
1.2 So What?	2
1.3 Positioning This Work in the Field	3
1.4 The Big Questions	11
1.5 Organisation of Thesis	12
1.6 Summary of Papers	12
<b>2 Mathematics and Computing</b>	<b>13</b>
2.1 Integrating Computing and Mathematics	13
2.2 Representation of Real Numbers on the Computer	18
<b>3 Theoretical Background</b>	<b>23</b>
3.1 The <i>Understanding by Design</i> Framework	23
3.2 Actor-Oriented Transfer	27
3.3 Sensemaking	29
<b>4 Methodology</b>	<b>35</b>
4.1 Iterative design process	35
4.2 The Interviews	36
4.3 Analysis	37
<b>5 The Path to Paper I</b>	<b>41</b>
<b>I How Computation Can Facilitate Sensemaking About Physics: A Case Study</b>	<b>45</b>
I.1 Introduction	45
I.2 Analytical Framework	46

## Contents

---

I.3	Methods . . . . .	46
I.4	Computational Sensemaking Case . . . . .	48
I.5	Discussion and Conclusions . . . . .	51
	References . . . . .	53
<b>6</b>	<b>The Path to Papers II and III</b>	<b>57</b>
<b>II</b>	<b>Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing</b>	<b>63</b>
II.1	Introduction . . . . .	63
II.2	Theoretical Perspective and Background Literature . . . . .	64
II.3	Methods . . . . .	68
II.4	Results . . . . .	76
II.5	Discussion and Conclusions . . . . .	104
	References . . . . .	110
<b>III</b>	<b>Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study</b>	<b>115</b>
III.1	Introduction . . . . .	116
III.2	Theoretical Framework . . . . .	117
III.3	The Big Ideas . . . . .	121
III.4	Methodology . . . . .	122
III.5	Initial Design . . . . .	125
III.6	Results, part I: Initial Implementation . . . . .	129
III.7	Second Design . . . . .	136
III.8	Results, part II: Second Implementation . . . . .	140
III.9	Final Design . . . . .	158
III.10	Discussion, Conclusion and Avenues for Future Research . . . . .	160
	References . . . . .	168
<b>7</b>	<b>Discussion and Conclusions</b>	<b>173</b>
7.1	Summary of Findings . . . . .	173
7.2	Limitations . . . . .	176
7.3	The Big Answers . . . . .	177
7.4	Implications for Teaching . . . . .	178
7.5	Implications for Future Research . . . . .	179
	<b>Bibliography</b>	<b>183</b>
	<b>Appendices</b>	<b>189</b>
<b>A</b>	<b>Tutorial 1: Rounding errors</b>	<b>191</b>
A.1	Final Version of Tutorial . . . . .	191
A.2	Code for Tutorial 1 . . . . .	193
<b>B</b>	<b>Tutorial 2: Taylor Expansions</b>	<b>195</b>
B.1	Final Version of Tutorial . . . . .	195

	B.2 Code for Tutorial 2 . . . . .	201
<b>C</b>	<b>Tutorial 3: Numerical Integration</b>	<b>205</b>
	C.1 Final Version of Tutorial . . . . .	205
	C.2 Code for Tutorial 3 . . . . .	209



# List of Figures

1.1	Current RUME research foci in the context of computing . . . . .	4
1.2	Sub-foci in Student Thinking and Learning . . . . .	5
1.3	Sub-foci in Issues of Equity . . . . .	6
1.4	Sub-foci in Teaching . . . . .	6
1.5	Sub-foci in Interdisciplinary Research . . . . .	7
1.6	Map of papers and their intersections with research foci . . . . .	9
1.7	Map of the theory landscape . . . . .	10
1.8	Map of papers and their intersections with research foci . . . . .	10
2.1	An unexpected rounding error in Python . . . . .	21
2.2	The practical limit for that rounding error . . . . .	21
3.1	The sensemaking process . . . . .	30
5.1	Output of the code in Code Sample 5.1 . . . . .	43
6.1	Distribution of tutorial versions across the final two papers . . . . .	61
II.1	Cross-domain connections . . . . .	74
II.2	Gina’s first expression . . . . .	78
II.3	Line connecting $x$ and $diff_1$ . . . . .	80
II.4	Gina’s second expression . . . . .	80
II.5	Benjamin’s first expression . . . . .	82
II.6	Benjamin’s second expression . . . . .	82
II.7	Benjamin’s third expression . . . . .	83
II.8	Benjamin’s third expression . . . . .	84
II.9	The output visible in the terminal window, showing two rounding errors . . . . .	85
II.10	The students’ initial results for $a = 0.75$ and $n = 100$ . . . . .	89
II.11	Rita’s equation for the remainder at $x = 1$ . . . . .	90
II.12	The endpoint of Rita’s first whiteboard calculation . . . . .	91
II.13	Output of the students’ program for $x = 1$ . . . . .	95
II.14	Plot of the remainder with $n = 17$ . . . . .	95
II.15	Rita’s calculation for $x = 0.5$ . . . . .	97
II.16	Output of the students’ code for $x = 0.5$ . . . . .	97
II.17	Result of using the number of terms in Figure II.16 . . . . .	98
II.18	Tests of the students’ own log function . . . . .	98
II.19	The drawing from Martin’s worksheet . . . . .	101
II.20	Replicating Program, as seen in Case A . . . . .	105
II.21	Improvement Cycle, as seen in Case B . . . . .	106

## List of Figures

---

II.22	Justified Improvement, as seen in Case B . . . . .	106
II.23	Justified Design, as seen in Case C . . . . .	107
III.1	Relative errors for different $a$ with $n = 10$ . . . . .	129
III.2	Relative errors for different $n$ with $a = 5$ . . . . .	130
III.3	Plots produced by the students' code in the first phase interview	132
III.4	Test case for the remainder function . . . . .	140
III.5	Rita's usage of logarithm rules on the machine representation of $x$	142
III.6	The remainder plot with Rita and Lena's initial choice of parameters	145
III.7	The endpoint of Rita's first whiteboard calculation . . . . .	147
III.8	Rita and Lena's revised parameter choices . . . . .	153
III.9	Rita and Lena's final, optimal choice of parameters . . . . .	156
7.1	Sophia's third sensemaking segment re-interpreted . . . . .	173
B.1	Test plot for self-assessment in Tutorial 2 . . . . .	197
B.2	Output of test program of log function . . . . .	199
B.3	Asymmetric remainder to be explained by the students . . . . .	201
C.1	Normal distribution function between -10 and 10 standard deviations	207
C.2	Normal distribution function between -100 and 100 standard deviations . . . . .	208
C.3	Relative error scaling with increasing number of steps . . . . .	208



# List of Tables

- 6.1 Overview of the MAT-INF1100 tutorials and where they fit into the course schedule . . . . . 59
- 6.2 Students and interviews included in the final two papers . . . . . 62
  
- II.1 Properties used for labelling connections. . . . . 72
- II.2 Connection labels. . . . . 73
- II.3 Connection patterns. . . . . 108
  
- III.1 Effects of changing  $x$  on  $\xi$  and the remainder . . . . . 154
- III.2 Versions of the tutorial . . . . . 158
  
- B.1 Functions for Tutorial 2 . . . . . 196
- B.2 Variables for Tutorial 2 . . . . . 196
- B.3 Remainder values . . . . . 200



# List of Code Samples

5.1	Python code illustrating the difference between two approaches to rounding: during and after the calculation (the author’s work, not the student’s). . . . .	43
II.1	The program from Tutorial 1, written by the first author. . . .	77
II.2	Lena’s translation of the equation in Figure II.12 to the code editor (for $x = 1$ ). . . . .	92
II.3	Lena’s first attempt at a Python solution. . . . .	93
II.4	Competing loop possibilities, combined. . . . .	94
II.5	Lena’s final version of the loop. . . . .	94
II.6	Martin’s implementation of the midpoint function. . . . .	102
III.1	Rita and Lena’s implementation of the function that calculated the remainder. They wrote everything except the first two lines, which were given in advance as scaffolding. The function name <code>errorterm</code> was in later versions changed to <code>remainder</code> , to better evoke the relevant mathematical concept in familiar terms. . .	141
III.2	Rita and Lena’s implementation of the logarithm function. They wrote the last two lines, the rest was given in advance. . . . .	144
III.3	Rita and Lena’s code to solve the equation computationally. . .	151
III.4	Equivalent code to Code Sample III.3, without using mathematical work. . . . .	152
A.1	The code given to students for tutorial 1. . . . .	193
B.1	The file <code>taylorlog.py</code> where students make their own log function.	201
B.2	The file <code>taylor_test.py</code> that plots the remainder. . . . .	203
B.3	The file <code>log_test.py</code> that tests the students’ log function. . .	204
C.1	The file <code>integrals.py</code> where the students integrate the normal distribution. . . . .	209
C.2	The file <code>plots.py</code> that generates plots of the relative error. . .	210



# Chapter 1

## Introduction

Let us begin with a brief comment on vocabulary: throughout this thesis, when I use the word *computing*, I will follow (Lockwood and Mørken, 2021) and take it to mean *machine-based* computing: "the practice of developing and precisely articulating algorithms that may be run on a machine" (p. 2).

To be even more precise, in this specific context this means that the students use computer programming (Python) to solve problems that are both mathematical and computational in nature: in solving these problems, one must consider both *what* is modelled by the computer and the specifics of *how* one implements the model.

### 1.1 Why?

Why study computing in mathematics and science education? The main reasons, in my view, are pragmatism, vision and caution.

Pragmatically, computing unlocks classes of problems that cannot be solved without computers and expands the scope of what students can do. Many problems that are difficult or time-consuming to solve analytically become tractable for students when computers and programming are available. With computing, the work students do gets closer to the real-world work of modelling and estimation that is done in mathematics and science every day. This is not to devalue analytical work, but simply to acknowledge its limits.

Taking the visionary view, computing is both hands-on and flexible. Allowing students to see and visualise the values that symbols represent and tailor their solutions to their knowledge and preferences may well result in a better understanding of the concepts. When students differentiate numerically, estimating the slope of curves in tiny steps, one could claim that they get closer to what differentiation *is* compared to analytical differentiation. Indeed, some of our students have asked their teachers whether the differentiation they do on the computer is in any way related to the rules-based algebra magic that is analytical differentiation. Finally, my own data strongly suggests that interesting things happen when we integrate computing in a mathematical or scientific context.

Though it is not the main focus of this thesis, there is also some reason for caution: If students learn that computational tools produce the correct answers as if by magic, and one simply has to take these answers on faith (Watters and Watters, 2006), we will not realise the aforementioned potential for understanding. That may have a lot to do with *how* we teach students computing, and how it integrates with other subjects. Also, we need to be careful that the barrier to entry does not become too high. If students struggle with the fundamentals of

## 1. Introduction

---

programming, using computing to get at complex understandings is probably going to be too much to ask for. Some of my data suggests that these issues may be related: when students do not know how to use what they have learned, they may be tempted to use the computer as a "black box" that spits out correct answers (Gravemeijer et al., 2017).

Currently, there is not an abundance of particular examples of how computing might be meaningfully integrated in mathematics and science education (Lockwood and Mørken, 2021). The education research community's need for such examples and the relatively long history of such integration at the University of Oslo, Norway, combine to form a compelling reason to study this and to do it here. In a sense, it seems long overdue to only now study what has been implemented in teaching for the last two decades, but I do not believe that science-based education dictates that one always has to test how something works in a controlled setting and then implement it, in that order. Even so, when collecting data, we did so with new teaching experiments that we designed based on classroom observations and teaching experience.

One question remains, however: why focus on computing in mathematics, of all the sciences? Mathematics is arguably the most fundamental science, in the sense that most if not all branches of science, technology, engineering and mathematics (STEM) make use of its tools to some degree. Therefore, difficulties in student understanding of mathematics have a tendency to make the teaching and learning of other subjects more difficult (Reddy and Panacharoensawad, 2017). If the availability of computational methods is changing the way we teach and learn mathematics, these changes will tend to spill over into other STEM subjects that rely on its tools. Thus, while I do not focus on computing in mathematics exclusively, I do regard it as a key part of computing in science.

### 1.2 So What?

We can imagine, then, that we have decided to study how computing impacts mathematics and science education, and that we have obtained some results. What is it good for?

My first goal is for my work to benefit students. Engaging students and having them do meaningful, authentic work while learning is important for them to understand the material and learn more than simple facts and skills (Wiggins and McTighe, 2005). Our quest, which is to identify the potential that computational mathematics and science represents, may therefore benefit the learning of STEM students in substantial ways if this potential is realised.

My second goal is for my work to benefit teachers. Many educators want to integrate computing into their teaching, but fear that students will use these tools as substitutes for learning mathematics and science. In particular, if students treat calculators and computers as "black boxes", there is the danger that the students will not be able to use these tools sensibly and flexibly, and certainly not modify them or build better ones. It is my belief that one need not sacrifice science and mathematics on the altar of computing, and I would like for this

work to be a contribution to educators who are wondering how to incorporate computing in their classes in a research-based manner.

My final goal is for my work to benefit researchers. While education research communities have provided substantial insight into how humans learn mathematics and computer science over the years, comparatively little work has been done on the intersection points of these disciplines.

Traditionally, mathematics and informatics have been organised as separate departments, and basic programming has been taught in dedicated classes separate from mathematics classes. With such an approach, students have barely been exposed to programming outside these dedicated classes, and the use of computing in science has been poorly motivated, leaving students with little experience on how to code in a scientific context (Stormo, 2009).

As this is changing, however, there will be both opportunities and higher demand for research on cross-disciplinary learning that incorporates mathematics and computing in a STEM context. This mirrors how computational methods are becoming part of the scientific disciplines to an increasing degree (Weintrop et al., 2016) and there are now opportunities for students to do authentic scientific work that involves computing. I would like my work to be of help to the researchers that set out along this path, by providing examples of what is possible in the computational classroom and how to bring it about.

In a word, the aim of this project is to support our *understanding*, whether we think of ourselves as students, teachers, researchers, or any combination of the three.

### 1.3 Positioning This Work in the Field

The field of Research in Undergraduate Mathematics Education (RUME) can be organised into several current research foci. One way to do this – by no means the only or ultimate way – was suggested by (Lockwood and Mørken, 2021), in the context of extending current work in RUME to include computing. Lockwood and Mørken identified four research foci which were particularly suited to include research that straddle the mathematics-computing boundary:

- Research on Student Thinking and Learning
- Research on Issues of Equity
- Research on Teaching
- Interdisciplinary Research

As Figure 1.1 suggests (and Figures 1.2 to 5 show more clearly), we are not aiming to cover *all* of the RUME community’s foci in a single thesis. The research presented here is focused mainly on Student Thinking and Learning, but Teaching and Interdisciplinary Research are also important features of the work.

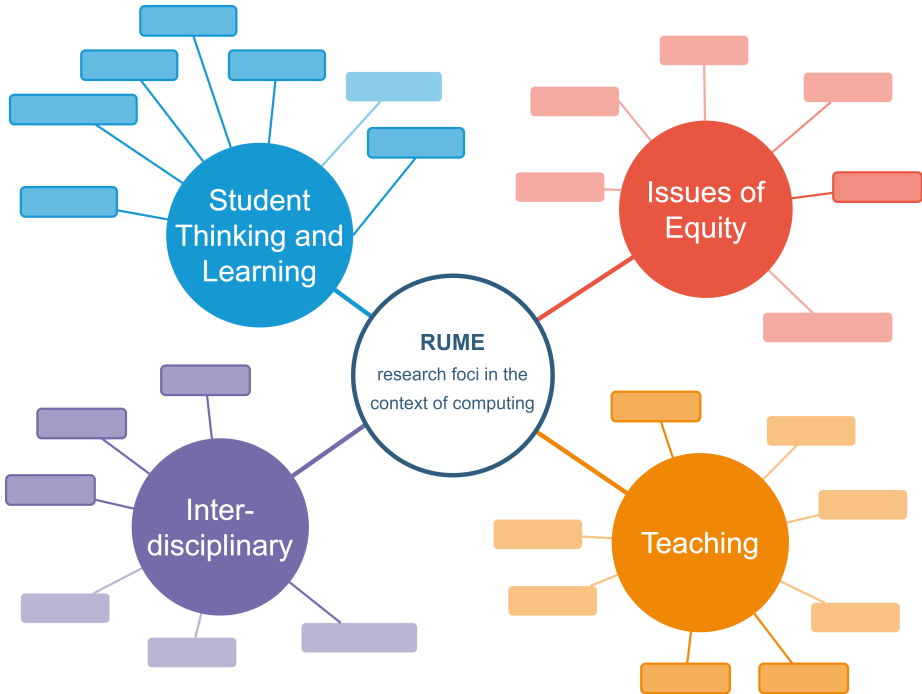


Figure 1.1: Current RUME research foci in the context of computing as outlined in (Lockwood and De Chenne, 2020), with the foci of the papers in this thesis highlighted. Each sub-field is detailed in Figures 1.2 to 1.5.

Student Thinking and Learning deals with how students learn and think about mathematical practices and concepts. How computing may interact with students' reasoning is important to uncover, and not only because the presence of computing in classrooms is growing. To realise the full potential in the interplay between computing and mathematics, it is vital to be aware of both affordances and hindrances to student understanding in learning environments that depend on computing. The sub-foci of Student Thinking and Learning that are relevant to this thesis are depicted in Figure 1.2.

An important subset of this research focus concerns *transfer*, the ability to make use of what one has learned outside the context in which it was learned. Indeed, one can claim that the concept of transfer is the very foundation on which institutions of education rest. What good, after all, is knowledge that cannot be employed outside the school or university setting? (Billett, 2013) We will elaborate on how we regard transfer later in this section.

In pedagogical terms, the work in this thesis springs from an individual cognitive perspective, as I take the *constructivist* view and study how the individual constructs knowledge from pieces of information. This does not mean that I have entirely ignored the *sociocultural* dimension, where learning is rooted



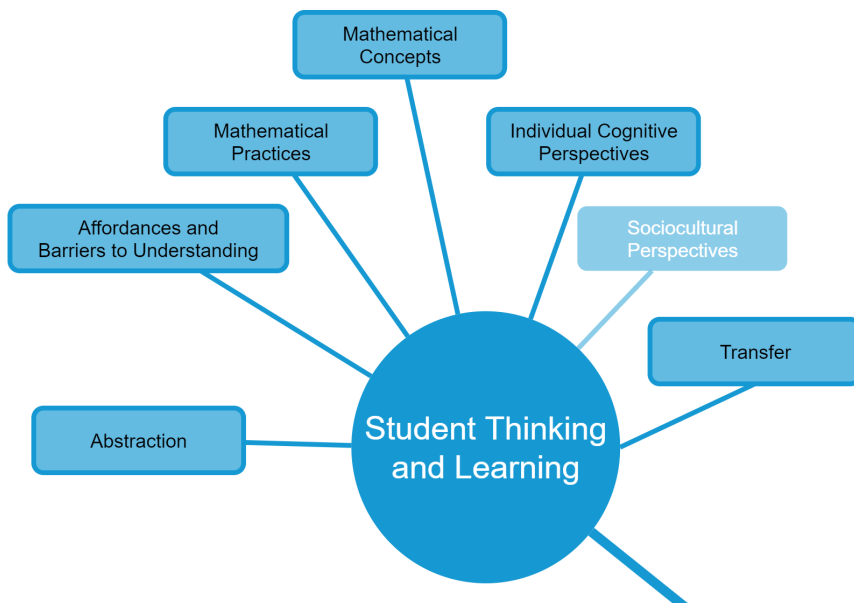


Figure 1.2: RUME sub-foci in Student Thinking and Learning, with the foci of the papers in this thesis highlighted.

in the social context. Even though I have not deliberately used it here, *social constructivist* theory incorporates both of these perspectives (Penprase, 2020), which at least demonstrates that they need not be incompatible.

Issues of Equity are important both inside STEM and elsewhere. However, as most of these issues affect an entire cohort of students over long periods of time, it was difficult to study them with the few students that we worked with for a single semester each. Nonetheless, I do address one such issue, namely ways to support student engagement in the design of the tutorials for papers 2 and 3 (see Figure 1.3).

Research on Teaching, as shown in Figure 1.4, can be divided into several categories. I do not focus on the instructors (including pre-service teachers), nor on department or program level issues, for the same reasons that I selected few Issues of Equity. I found student engagement, task design and the specifics of integrating mathematics and computing to be a better fit to our scope.

The final category is Interdisciplinary research, shown in Figure 1.5. Cross-disciplinary integration of computing proved difficult to fit to our scope, and terminology had to take a back seat to make room for the more salient issues of generalising, problem solving, concepts, symbols, and operations.

Some symbols, like the equality symbol ( $=$ ), take on different meanings in each of the two *domains* of mathematics and computing. In Python, its role is to assign a value to a variable. In mathematics, it can play the similar role of defining a symbol, but not necessarily assigning a data type to the value or

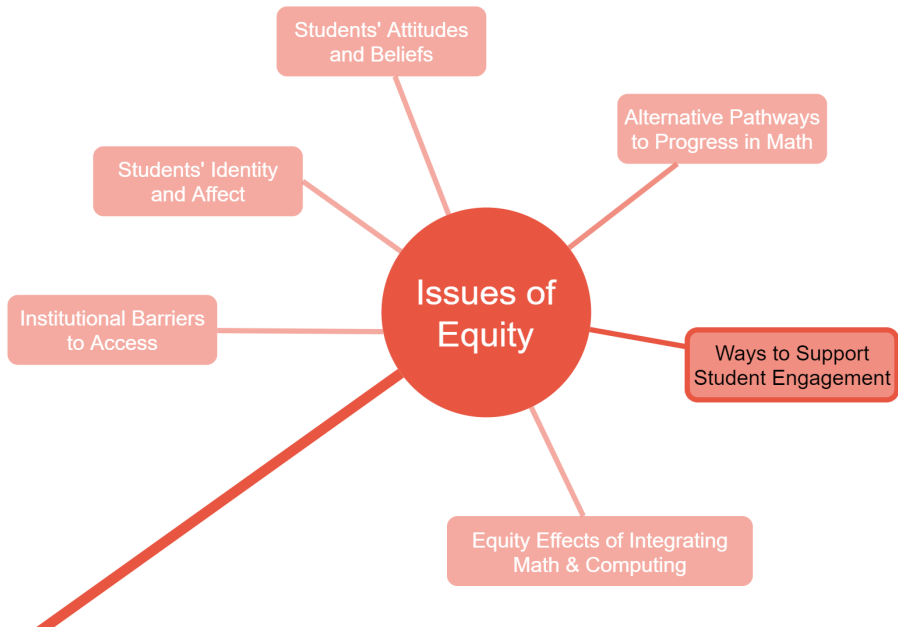


Figure 1.3: RUME sub-foci in Issues of Equity.

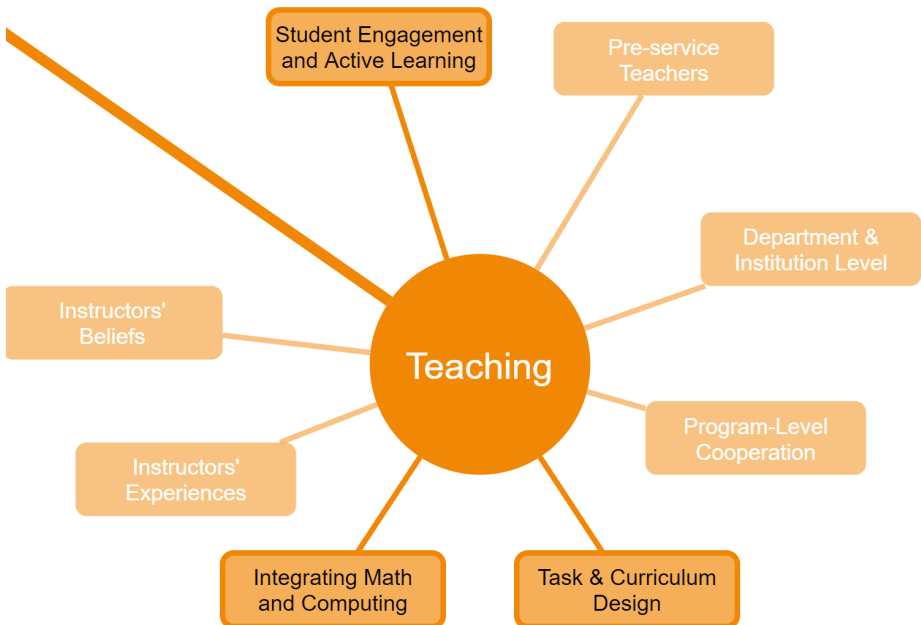


Figure 1.4: RUME sub-foci in Teaching.

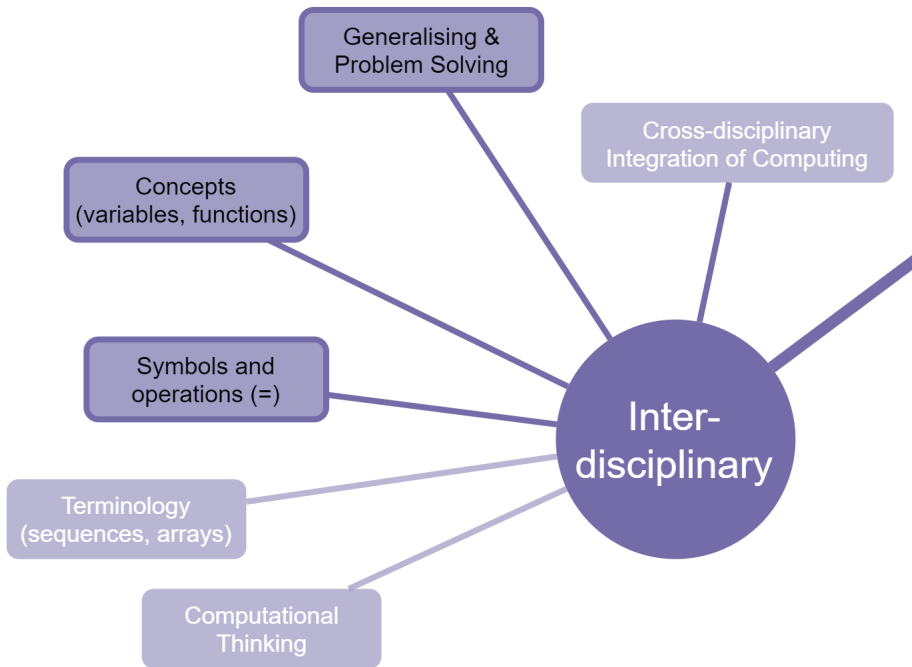


Figure 1.5: RUME sub-foci in Interdisciplinary Research.

storing it somewhere in a computer’s memory. Depending on context, = can also be used to equate one quantity (which may be more complex than just a symbol) to another and argue mathematically what follows from that – this is often called an *equation* instead of a definition. A different Python operator (==) can determine whether two quantities are equal or not (returning True if they are and False otherwise), but the mathematical reasoning of what follows from this is missing.

Variables and functions have different, but related, meanings in mathematics and computing (Knuth et al., 2011; Wright et al., 2013). A mathematical function can be seen as a mapping between sets: input to output. Although many different inputs may produce the same output, the converse is not true<sup>1</sup>. Computational functions do not respect this constraint. A classic example is `random.random()`, a Python function which returns a different pseudorandom number between 0 and 1 each time it is called. Additionally, computational functions can do things that mathematical functions cannot, such as printing some text to screen, and may not even produce (return) any output at all.

<sup>1</sup>For instance, if  $x$  can be any real or any complex number,  $f(x) = 42$  is a perfectly legal function definition in mathematics.  $f(42) = x$  is not – such a definition would implicitly constrain  $x$  to be just one number, and the "definition" does not even tell us which one it is.

## 1. Introduction

---

Regarding mathematical functions as a true subset of computational functions would be too simplistic, however. Take for example the Dirichlet function (Kudryavtsev, 2017), which is defined on the interval  $0 < x < 1$ , so that  $f(x) = 0$  if  $x$  is an irrational number, and  $f(x) = 1$  if  $x$  is rational. Determining whether or not a number is rational is no easy task for a computer. A rational number could turn out to contain repeating sequences of  $10^{100}$  digits or longer, so one would need infinite resources to determine this with certainty. This is an example of a non-computable function that is nonetheless mathematically useful, for instance in measure theory.

Since Seymour Papert introduced the concept of *computational thinking* (Papert, 1993), several studies have tried to apply it to the mathematics classroom (Hickmott et al., 2018; Sinclair and Patterson, 2018). However, it has been rather challenging for the research community to settle on a working definition of the concept, as noted by Shute et al., 2017. Additionally, (Tedre and Denning, 2016) address the dangers of, among other things, exaggerating the potential for transfer from computational practices to other areas of science: computational problem-solving skills do not *automatically* transfer to other domains.

As (Lockwood and Mørken, 2021) point out, many studies that examine claims such as these take a rather traditional view of transfer, where transfer is measured by student performance on a "transfer task" that is pre-defined and as such may miss the ways in which novice learners (attempt to) generalise their knowledge<sup>2</sup>. Nonetheless, I find the issues mentioned above problematic enough to not use computational thinking as a theoretical lens in this thesis or its papers.

This thesis is built around three academic papers, and the sub-foci each paper addresses is shown in Figure 1.6. The first paper is centred on the concept of sensemaking, which intersects all four research foci. The second paper is about Student Thinking and Learning, while the third is a Teaching paper.

Figure 1.7 maps out my theory landscape, showing the theoretical framework for each paper and the ways in which these theories connect to each other. Each paper represents an important step on my journey through this map, and a timeline of these milestones is shown in Figure 1.8. In this chronological representation of the work, I also point out how the papers are related.

I refer to Section 3 and the papers themselves to elaborate on each theoretical focus, but we have time for a brief summary of the ways in which they are connected, as depicted in Figure 1.7. In short, *sensemaking* is a cyclical process by which students realise there is something they do not understand and try out various ways of resolving the conflict in their understanding (Odden and Russ, 2018). This process may be conceived to consist of many individual *connections*: realising that two pieces of knowledge are related, similar, or dissimilar. These connections are central to the theory of *actor-oriented transfer* (AOT) as described in (Lobato, 2012), where transfer is defined as any generalising activity, regardless of correctness, by the learners. On the basis of this, I

---

<sup>2</sup>We shall return to this issue shortly, in Section 3

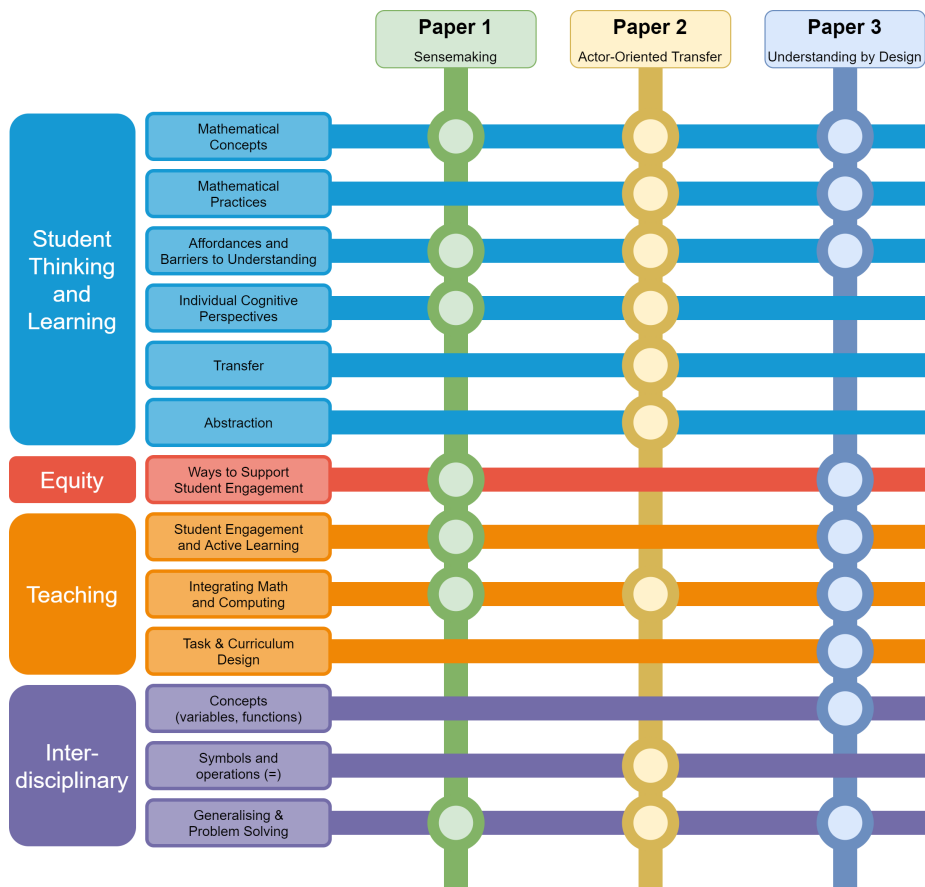


Figure 1.6: Map of papers and their intersections with the research foci from Figure 1.1.

relate these concepts by granularity: sensemaking describes the process of understanding as a whole, whereas AOT looks at it at a more fine-grained level.

The use of the term *understanding* in the previous paragraph suggests that both of these theories may also relate to the *Understanding by Design* (UbD) framework put forward in Wiggins and McTighe, 2005, and it may not come as a surprise that I believe this to be the case. Whereas AOT looks at connections from the learners' point of view, UbD takes the teacher's or teaching designer's perspective. Essentially, UbD defines the understandings that we seek and promotes clarity on learning goals and learning activities, while sensemaking is the process on the students' part that leads to these understandings.

These relations can be summed up as follows:

- Sensemaking consists of many connections that we can study with AOT.

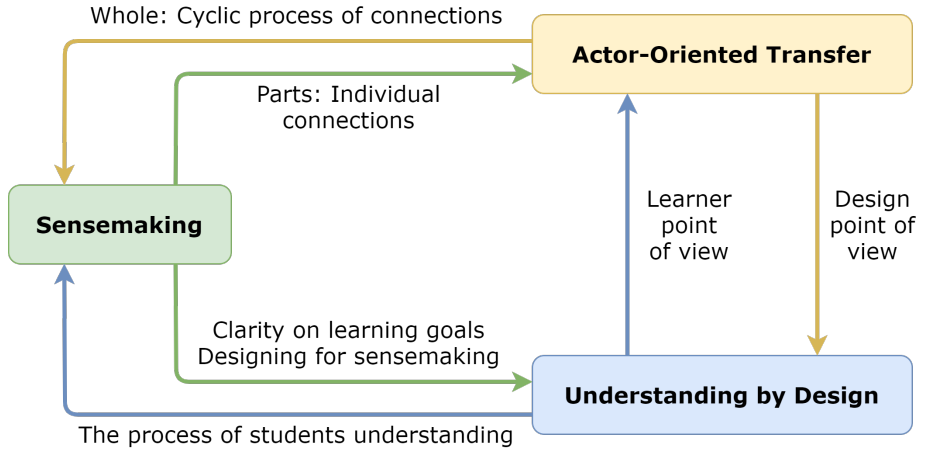


Figure 1.7: Map of the theory landscape, showing connections between the theoretical frameworks.

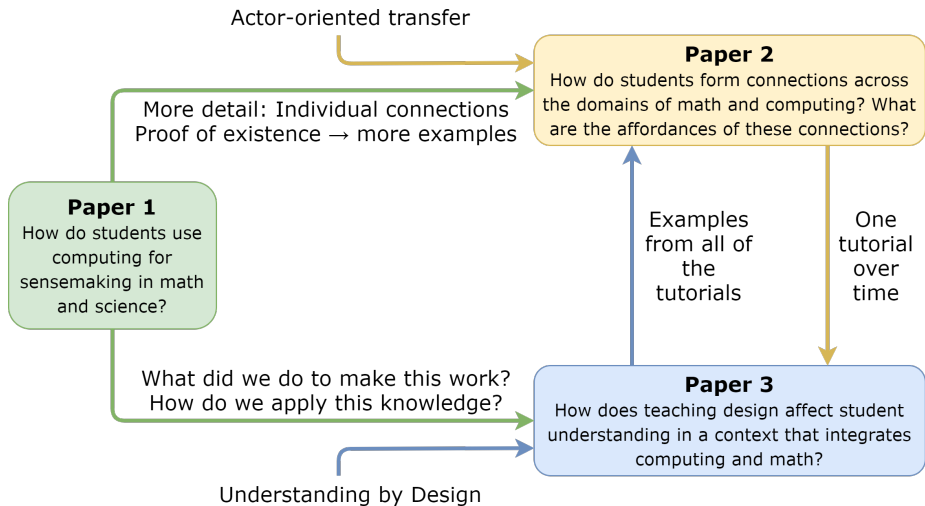


Figure 1.8: Timeline of the papers, showing the paths I traversed through the theory landscape presented in Figure 1.7.

- Sensemaking and AOT describe what happens from a student perspective; UbD brings the teacher perspective into the mix: how do we enable and support students' understanding?

My first paper examines sensemaking in a physics context, where mathematics and computing are both strongly present. From that proof of existence, I wanted to (a) provide more examples of this happening, and (b) examine the connections between mathematics and computing in detail. This led to my second paper, in which I provide several examples of integrated teaching of mathematics and computing, as well as using AOT as a theoretical lens to investigate the process in greater detail.

In parallel to this, having demonstrated that a task I designed for such an integrated context could produce sensemaking in students, I was also curious about the implications for teaching. How could I produce more learning experiences like this? Which design principles best bring about sensemaking and understanding? To answer these questions, I designed three tutorials using the UbD framework and investigated the results.

Examples from all three tutorials made it into Paper II, so that I had as complete a picture of the students' connections as my data allowed. Paper III, on the other hand, investigates one of the tutorials in greater detail, to examine what happened as a result of our design choices and why that happened. While all the tutorials were designed for a first-semester course at the University of Oslo, we have made them available for a wider audience (see Sections A to C), and believe they could be useful in many higher education contexts with little or no modification.

### 1.4 The Big Questions

Looking at my project as a whole, my work can be summarised as attempts to answer the following questions:

- How do the students themselves integrate science, mathematics, and computing in the context of representing real numbers on the computer?
- What are the resulting affordances for learning?
- How does the design of learning activities support or hinder this integration?

As cross-disciplinary research in this field is sparse (see Section 2) my work in all three papers takes the form of exploratory case studies. I went looking for interesting ways in which the students reasoned across domains, what that afforded them, and which design principles allowed this to occur. Therefore, I will not be able to answer these questions comprehensively. Nonetheless, these questions guided my inquiries and helped with the selection and analysis of the cases I present here.

### 1.5 Organisation of Thesis

The remainder of the thesis is organised as follows:

- Section 2 elaborates on the context of my research. What does it mean to integrate mathematics and computing, how is it done at the University of Oslo, and more specifically, what do we mean by the computer's representation of real numbers?
- Section 3 lays out the theoretical landscape which I mapped out in Figure 1.7. To be able to describe the connections between these theories, this section overlaps somewhat with the theory sections in the three papers.
- Section 4 presents the methodology that is common to all three papers, again with some overlap with the methodology sections of the papers themselves.
- Sections 5 and 6 tells the story of my research, as represented by Figure 1.8. I briefly motivate each paper and explain our choices along the way. These sections are presented chronologically, so the reader can read the papers in the order that they were conceived.
- Section 7 is dedicated to a summary and discussion of my results, in which I also seek to answer the questions of Section 1.4.
- Finally, the tutorials (including Python code) that I designed for the final two papers can be found in the appendix (Sections A to C).

### 1.6 Summary of Papers

**Paper I** demonstrates how a student used a computational representation of a problem as a resource in sensemaking.

**Paper II** uses actor-oriented transfer to describe four ways in which students connected the domains of mathematics and computing.

**Paper III** shows how we used the Understanding by Design framework to iteratively design a tutorial and the lessons learned from that process.



## Chapter 2

# Mathematics and Computing

In the beginning, computers were designed to do mathematics. Ada Lovelace is recognised as the author of the world's first computer program, one that described an algorithm for calculating Bernoulli numbers (Carlucci Aiello, 2016). Algorithm design was considered part of mathematics for a long time afterwards. In the beginning, the hardware was highly specialised, but around 1940, the quest for nuclear fission resulted in machines that were both general and flexible, and eventually rather fast. This led to a renaissance of the numerical methods for approximation known since the times of Newton. Finally, in the late 50s, computer science emerged first as a term, then as a field in its own right.

This separation between mathematics and computer science has also been reflected in an educational context, at least up until the time that computer hardware became common enough to be used in schools. When I speak of integrating the two domains of mathematics and computer science, in a sense I mean re-integrating two fields with a common origin and many connections between them, merging the results of this parallel evolution where it is appropriate to do so.

Section 2.1 will discuss this integration in broad terms, provide a literature review of other studies in the field, and position my own work relative to what has been done before. I will similarly position the University of Oslo context in an international perspective, using research literature to describe similarities and differences.

Section 2.2 will focus on one aspect of computing in science that has been central to all three papers in this thesis: the representation of real numbers on the computer. It is well known that the computer cannot represent all real numbers, not even most of the rational ones (Mørken, 2017). But instead of considering this to be only a limitation of the hardware, is it possible that the computer's model of the real numbers can teach us something about the real numbers themselves, and vice versa?

## 2.1 Integrating Computing and Mathematics

One of the first to discuss computing in a mathematics education context was Alan J. Perlis, a central founding figure of computer science as a discipline. He argued that the way one teaches approximations and algorithms in relation to discrete and continuous analysis matters greatly. If done in a sensible way, both students and the fields themselves would be better off for it (Forsythe et al., 1970)<sup>1</sup>.

---

<sup>1</sup>The other authors' contributions to the same paper are well worth reading, as it offers a contemporary perspective on what the overlaps and differences between mathematics and

## 2. Mathematics and Computing

---

The perceived value of computing in mathematics education was popularised by the work of Seymour Papert (Papert, 1993), who proposed that the computer offered unique affordances for students learning mathematics. Using the *Logo* programming language designed for this purpose, Papert studied children's discovery of algorithmic ways of thinking about geometry through use of the computer. The classic example of this is children writing programs that direct a Turtle across the screen, tracing out geometrical shapes in the process. Papert found that in this setting, children were motivated and able to construct knowledge: powerful abstract ideas became concrete for them as they worked with computers that afforded meaningful representations of these ideas.

Integrating mathematics learning with computing is best illustrated by a counterexample. In a *disintegrated* design, the students learn to code in a context that is mathematical only by accident. For instance, creating a webpage for flight booking with a database back-end would exemplify such a disintegrated learning activity. Presupposing then that students know how to program, instructors in future courses might then simply refer students to the relevant math software libraries and focus on the *application* of these presupposed skills.

The issue with this approach is related to transfer of learning: the context in which learning takes place has an impact on both the learning itself and the potential for transfer to other contexts (Billett, 2013). Hence, learning to code in a non-mathematical context and separately learning mathematics does not equate to students being able to use code effectively *in* mathematics.

(Buteau et al., 2020) point to one feature of what this may entail: To articulate a mathematical process in a programming language, one translates into the language what one would do by hand. To do this, one must realise that the code can indeed work in a similar manner as one does by hand, which is neither self-evident nor independent of what kind of mathematical work one engages in. Integrating coding in mathematics then entails supporting the students in learning, in the words of Buteau et al., "to transform a programming technology into a rich 'mathematical instrument' enabling him/her to [participate] in programming-based mathematical work" (p. 1029).

On the other hand, the way the computer solves mathematical problems is more often than not very different from the way a human would go about it. To quote an example from (Mørken, 2017) in the context of solving equations:

This illustrates how an experienced equation solver typically works, always looking for shortcuts and simple numbers that simplify the calculations. This is quite different from how a computer operates. A computer works according to a very detailed procedure which states exactly how the calculations are to be done. The procedure can tell the computer to look for simple numbers and shortcuts, but this is

---

computer science are. Forsythe, for instance, asks (and attempts to answer) why mathematics and computer science should be considered different fields at all. It is somewhat amusing that 50 years later, we are looking back and arguing for the benefits of integrating the two disciplines in education.

usually a waste of time since most computers handle fractions just as well as integers. [...]

These simple examples illustrate that when (experienced) humans do computations they try to find shortcuts, look for patterns and do whatever they can to simplify the work; in short, they tend to improvise. In contrast, computations on a computer must follow a strict, predetermined algorithm. A computer may appear to improvise, but such improvisation must necessarily be planned in advance and built into the procedure that governs the calculations. (pp. 7-8)

In short, this suggests that computing and mathematics and computing need not be taught or learned separately, that students can replicate (and perhaps extend) their mathematical work by computing, and that the ways in which they do this may differ from what they are used to.

The big questions in Section 1.4 then motivate the following question: How will the differences in how students approach mathematics computationally, as opposed to analytically, manifest themselves in the students' understanding of mathematics? The answer, one suspects, is that it may depend on how one goes about it.

### **2.1.1 A brief literature review of computational tools in mathematics education**

This literature review is all but identical to the one that appears in my third paper (Paper III). As it is relevant to all my work, however, it bears repeating here rather than having the reader wait until near the end to see it.

There are numerous examples in the mathematics education literature of different computational tools being implemented as part of learning activity designs in university mathematics. (Dimiceli et al., 2010) showcase a design experiment where the symbolic Computer Algebra System (CAS) features of the WolframAlpha app were used as an asset in an introductory calculus course. Compared with other CAS software, they found that it had several advantages, although processing power was a limitation. A similar design experiment described in (Caglayan, 2016) demonstrates ways to use the *GeoGebra* dynamic software to visualise Riemann sums, allowing students to visualise and discover important properties of these sums.

Beyond showcasing that designs incorporate these technological tools, there are also studies that investigate the relationship between task design and students' use of them. (Olsson, 2019) comparatively investigated two designs that used *GeoGebra*, in this case a task involving functions designed for schoolchildren in grade 7 to 9. That study, interviewing students in pairs, found that students who were encouraged to explain their thinking performed better overall than students who were encouraged to follow a set of written instructions.

There are also examples of software being designed specifically for educational purposes. One such example is *Grid Algebra* (Hewitt, 2016), a software designed

for learners as young as 9-10 years old to visualise the four basic arithmetic operations as movements on a grid when solving linear equations. The software called *Configure* (Greenstein, 2018) similarly lets younger students visualise and conceptualise topological equivalence.

In addition to using pre-existing software and writing dedicated software for educational purposes, there is a third option: having students write or modify computer programs written in a generic programming language. An integrated approach then demands that these programs are written in a mathematical context. One example of this is (Lockwood and De Chenne, 2020), in which students related combinatorial counting problems of different types to corresponding conditional statements in Python programs. This resulted in a reinforcement of conceptual understanding in an area students traditionally have difficulties with, and this reinforcement was attributed to the computational setting. My work in this thesis aims to provide more examples of this.

Another example is the design of a project (Ramler and Chapman, 2011) where students statistically analyse whether players' missed notes in the *Guitar Hero* video game are randomly distributed by writing code in R. In the process of analysing complex data, students would gain hands-on experience using statistical concepts to test their hypotheses (and generally find that the randomness of missed notes depends on the skill level of the player and the difficulty of the song being played). Unlike Lockwood and De Chenne, Ramler and Chapman focus mostly on their design and less on how the setting may influence the reinforcement of concepts. Nonetheless, their design resembles that of the previous example and belongs in the same category.

Like the two examples just mentioned, our work is focused on university students using the Python programming language in a mathematical setting (this context is described more closely in the following section). In practice, that means our students use Python programming to articulate, visualise, investigate, and solve mathematical problems. Because the setting in which we collected data is unique, I will now focus on the particular integrated approach at the University of Oslo and situate this program within the literature.

### 2.1.2 The University of Oslo approach in perspective

Parts of this subsection also appeared in my second paper (Paper II).

All my studies are done in the context of undergraduate mathematics and science students learning mathematics and computing at the University of Oslo, Norway. Since the *Computing in Science Education (CSE)* initiative was introduced in the early 2000s, the first semester for students in mathematics, physics and electronics have consisted of three courses:

- A traditional *Calculus* course.
- A hybrid course focusing on *Modelling and Computations*, where calculus concepts are discussed in the context of computational algorithms and computer hardware. My last two papers concern the development of and research on a set of tutorials that I designed for this course.

- An *Introduction to Programming with Scientific Applications* course where students learn basic Python programming and implement the algorithms from *Modelling and Computations* in a scientific setting.

Typically, concepts are taught in a staggered approach, appearing first in Calculus, then discussed further in *Modelling and Computations* the week after, and in the third week becoming the topic of *Introduction to Programming with Scientific Applications* (Malthe-Sørenssen et al., 2015). A typical example is differentiation. After analytical differentiation has been covered in Calculus, algorithms for numerical differentiation appears in *Modelling and Computations* the next week, and the students implement these in the programming course the week after that. This setup can be interpreted as a realisation of the potential pointed out by Alan J. Perlis in (Forsythe et al., 1970), who argued that a well-integrated approach made sense in introductory courses like these.

It should be mentioned that in the early days of the CSE initiative, the programming course was decoupled from mathematics (much like our example of a disintegrated approach), and the students were not pleased about it. After the introduction of the current programming course in 2007, where students learn programming basics in a scientific setting, the trend in feedback from students has been that they are impressed with how well integrated computing has become in the first semester.

While this context of teaching mathematics and computing is to the best of our knowledge fairly unique, (Buteau et al., 2016) provide an example of a similar approach. In Brock University, Canada, they do not coordinate across informatics and mathematics courses in the same way. They do, however, dedicate a set of courses where students can use knowledge from both in an integrated fashion after learning programming and mathematics separately in earlier semesters.

In their paper, the authors present a case study of a single student spanning three semesters and the student's work on 14 assignments in the *Mathematics Integrated with Computers and Applications (MICA)* courses. These assignments were connected to many different mathematical topics, and the authors focused on the student's learning experience across these semesters. They concluded that the student meaningfully engaged with mathematics and a constructionist<sup>2</sup> type of learning. This study differs from my work in that they investigate courses designed from the ground up to provide these experiences – my tutorials, on the other hand, were designed to fit into and complement the pre-existing *Modelling and Computations* course.

Going beyond the undergraduate context, there are a higher number examples of studies exploring the interplay between mathematics and computing for younger students (grades 5 to 8 in these examples): (Lavy, 2006) used Logo to investigate the different types of mathematical arguments such students construct when working in a computerised environment. Lavy characterised four types of mathematical arguments that the students constructed in this setting.

More recently, Benton and colleagues designed interventions using Scratch to have schoolchildren reason with geometrical concepts and the values assigned

---

<sup>2</sup>Building knowledge structures (Papert, 1993).

## 2. Mathematics and Computing

---

to digits due to their positions in a number (Benton et al., 2017; Benton et al., 2018), demonstrating how this allowed the students to engage with challenging ideas in meaningful and generalisable ways. DeJarnette also used Scratch to investigate the difficulties that schoolchildren face when trying to understand the meaning of symbols and how they fit together DeJarnette, 2019, finding that young students were able to create their own representations that, while less detailed than those of experts, were nonetheless meaningful to their creators.

Going forward, the recent change in curriculum for Norwegian schools (UDIR, 2020) integrates algorithmic thinking into mathematics, taking a cue from the CSE initiative. The result of this will be that from 2023 and onwards, the majority of Norwegian first-year undergraduate students will have previous experience with using computer programming to do mathematics. In this context, how higher education will be able to meet the needs of these students is likely to depend on the level of understanding (see Section 3.1) that these students bring with them. Will they see mathematics and computing as separate, or interconnected? Will they have mainly procedural knowledge, but require more conceptual knowledge to tie it all together?

Whatever the answer, it is evident that Norwegian institutions of higher education must rise to the challenge, however it will be defined: to provide coherence and understanding of the skills students learn in high school, or to build on existing understanding in ways that are meaningful to the learners. Existing integrated learning environments like those at the University of Oslo will have to adapt, and programs that have not yet incorporated computing, or taught it separately from mathematics may have to be redesigned in more fundamental ways to be able to leverage the knowledge and meet the needs of the next generation of students in mathematics and science.

It should be noted that integrating computing with mathematics is not the only way to teach computer science in schools. For instance, it has been proposed (Connor et al., 2017) that schools in Scotland should introduce a curriculum not explicitly connected to mathematics. Whether the fields of computer science and mathematics *should* be integrated is an interesting and complex question, and I will not attempt to answer it with certainty in this thesis. I do, however, hope that my investigations of the integrated approach will prove useful to those that seek to answer questions like these.

### 2.2 Representation of Real Numbers on the Computer

This section goes into more detail on how the computer represents real numbers. Readers unfamiliar with the topic, be warned: we are about to get rather technical. The essence of what follows is that I explain the difference between how numbers are *actually* represented on the computer and the simplified version of it we exposed the students to in my third paper. This simplified version corresponds to the standard form of binary numbers, however, and there exists a Python function that returns this form, so while not entirely accurate, this version of what the computer does is still useful. In the process, I will also

discuss rounding errors in more detail, and explain why they are inevitable for most real numbers on the computer.

An important aspect of translating by-hand mathematics into code is being aware of the limitations of representing real numbers on a computer. Having typically 64 bits (binary digits) available to represent a real number, the computer will be unable to represent every number exactly, and this inability imposes hard restrictions on the relative accuracy of this representation, as well as on the results of calculations involving real numbers (Mørken, 2017).

Computers are built for working with binary numbers consisting of zeros and ones, and perform calculations most efficiently, if not most accurately, in this number system<sup>3</sup>. The standard representation of a real number  $x$  is the *floating-point*<sup>4</sup> representation as specified in the **binary64** standard (“IEEE Standard for Floating-Point Arithmetic”, 2019):

$$\tilde{x} = (-1)^s \cdot m \cdot 2^{e-(2^{10}-1)}$$

where:

- $s$  is the sign of the number (1 binary digit, or *bit*)
- $m$  is the *mantissa* (53 bits in 64-bit *double precision*)
- $e$  is the *exponent* (11 bits in double precision)

An observant reader will notice that  $1+53+11 = 65$ , which seems too much for a 64-bit format. As it turns out, any normalised non-zero binary number has 1 as its first significant digit. Hence, we can represent 53 bits of information using only 52 bits in memory when we include this *hidden bit*<sup>5</sup>. Therefore, one implicitly assumes that this digit is present and do not store it in memory, so that the mantissa is interpreted as the digits following the radix point<sup>6</sup> in the number  $1.b_{52}b_{51} \dots b_1$ , where  $b_i$  is bit number  $i$  in the 52 bits of  $m$ . If we interpret these 52 bits as an integer, we obtain the 53-bit mantissa by

$$m = 1 + \frac{b_{52}b_{51} \dots b_1}{2^{52}}$$

---

<sup>3</sup>Relatively efficient encodings that allow computers to work with decimal numbers do exist and are used in calculations where accuracy and minimal rounding error in the results are considered crucial, such as finance. One example is the *densely packed decimal* (DPD) representation, summarised in (Cowlshaw, 2002). These representations are still limited in accuracy, and they are slower in general (Anderson et al., 2009). Thus, the trade-off between accuracy and efficiency that our students faced in our research interviews is also something that concerns experts.

<sup>4</sup>As opposed to *fixed-point* representations, where one has a given and unchangeable number of bits to the left and right of the radix point.

<sup>5</sup>The trade-off is that if all the bits are zero, the number does not represent zero unless  $e = 0$ . In this special case, the first significant digit is assumed to be zero, meaning that we give up the ability to represent numbers with mantissa  $2^{-1023}$  in order to make way for the arguably more useful . Similarly  $e = 2048$  can be taken to represent infinity (Muller et al., 2010).

<sup>6</sup>More commonly known as the "decimal point", I avoid this usage because it implies that we are using decimal numbers. I could have used "binary point", but this term is rather uncommon in colloquial language.

## 2. Mathematics and Computing

---

As a result, we have

- $1 \leq m < 2$  (not an integer)
- $0 \leq e < 2^{11}$  (integer)

For educational purposes, this is rather technical. It may instead be useful, to express  $m$  and  $e$  using the derived properties:

- $M = m \cdot 2^{-1} \quad 0.5 \leq M < 1$
- $E = e - (2^{10} - 2) \quad -1022 \leq E < 1025$

Note that I have added 1 to the expression for  $E$  compared to the original representation to compensate for halving the mantissa. This choice of  $M$  reflects the standard form of exponential notation, which is what the Python function `frexp()` returns. This transformation results in a simplified representation, which I made use of in Papers II and III).

$$\tilde{x} = (-1)^s \cdot M \cdot 2^E$$

For a 64-bit binary number, the precision is determined by the 53 bits available to  $M$ : there is no error in  $s$  or  $E$ . Using Lemma 5.21 in (Mørken, 2017), the *relative error* is then bounded by

$$\left| \frac{x - \tilde{x}}{x} \right| \leq 2^{-53} = 1.110223 \cdot 10^{-16}$$

meaning that 53 bits in the mantissa allows the real number  $x$  and its floating-point representation  $\tilde{x}$  to have roughly 15 or 16 digits in common (if the error in the 16<sup>th</sup> digit turns a 9 into a 0 or vice versa, the 15<sup>th</sup> digit is also affected). Beyond that, we are bound to be victims of *rounding errors* for most real numbers, including every irrational number. When these numbers are used in calculations, there errors can be magnified considerably if we are not careful.

So far, we have looked at the computer's finite-precision model of the real numbers, which is quite different from the real numbers themselves. For instance, the mapping between the models is not one-to-one: given that we can pack infinite information into a real number, meaning that behind each floating-point number the computer can represent, an infinite number of real numbers may be hiding. However, there may be advantages to the way computers model real numbers as well, not only limitations.

As (Mørken, 2017) demonstrates, one can use the mathematics of real numbers to gain insight into the computer model. It turns out that rounding errors are not arbitrary, unpredictable quantities that we can only model statistically. In theory, it is possible to predict when rounding errors will occur. To use an example that I employed in the first version of Tutorial 1, I asked the students to print the result of the calculation  $2.2 \cdot 55$  in Python and explain the result (Figure 2.1):



```
>>> print(2.2*55)
121.00000000000001
```

Figure 2.1: An unexpected rounding error in Python.

In the normal (decimal) arithmetic<sup>7</sup>, it would seem absurd that this calculation would produce anything other than the precise value 121.0 (the integer 55 gets converted to the floating-point number 55.0 during the calculation). And at first glance, 2.2 seems to be a good fit for binary calculations as well. Unfortunately, the decimal part  $0.2 = \frac{1}{5}$  which only has one decimal digit turns out to be endlessly repeating in binary:

$$0.2_{10} = 0.001100110011\dots_2$$

Our students experimented with this unexpected result, discovering that the rounding error occurs when 2.2 is multiplied with integer values larger than 42 (Figure 2.2). The original rounding error is small enough that Python gets all the digits correct when displaying the result as a decimal number, but when it is magnified by multiplying with a large enough number, the error becomes visible.

```
>>> print(2.2*43)
94.60000000000001
>>> print(2.2*42)
92.4
```

Figure 2.2: The practical limit for the rounding error in Figure 2.1 to appear, found by students that I interviewed.

Another way that students may benefit from using mathematics to understand numbers in the computer is related to my third tutorial (Section C). Here, the students discover that when you increase the number of terms in the Riemann sum of a numerical integration, the accuracy grows worse due to the computer's limited precision. In this tutorial, the students integrate the standard normal distribution. This amounts to adding a large number of function values, and once past the peak one adds very small values to a comparatively very large, accumulated value. Prior to addition, the numbers are converted so their exponents are the same, with the result that the smaller numbers will be truncated and contribute very little, if anything, to the sum (Mørken, 2017).

---

<sup>7</sup>The numbers in Figure 2.1, although represented as decimal numbers were actually calculated with binary floating-point arithmetic, as per the rules of the *binary64* standard. If they had instead been calculated using decimal arithmetic like that of DPD (see footnote 2.2), we would not have gotten the same result.

## 2. Mathematics and Computing

---

Even as we use mathematics to understand the computer's model of real numbers, from a teaching point of view one can also go in the opposite direction and claim that working with floating-point numbers can increase students' understanding of true real numbers.

One example is the fundamental property that one can always find a sequence of rational numbers that *converges* to a given real number. On the computer, the most practical way to do this is to construct a sequence of fractions with powers of 2 in the denominator, which will approximate the number to arbitrary precision:

$$0.2 \approx \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^7} + \frac{1}{2^8} + \frac{1}{2^{11}} + \frac{1}{2^{12}} \cdots = \sum_{i=1}^{\infty} \frac{1}{2^{4i-1}} + \frac{1}{2^{4i}} = \sum_{i=1}^{\infty} \frac{3}{16^i}$$

This convergence and the resulting *completeness* of real numbers is not something one should take for granted. It is connected to the concept of limits in that we can get arbitrarily close to the target number given sufficient memory, even if the number itself is irrational or impossible to represent using only powers of two<sup>8</sup>.

Another example is my work in Tutorial 2 (Section B), which has students approximate the natural logarithm using the Taylor expansion:

$$\ln x = \ln (M \cdot 2^E) = \ln M + E \cdot \ln 2 \approx \ln a + \sum_{i=1}^n \left[ -\left(\frac{1}{i}\right) \left(1 - \frac{M}{a}\right)^i \right] + E \cdot \ln 2$$

for carefully chosen values of  $a$  and  $n$  when we presume to know the value of the constants  $\ln 2$  and  $\ln a$  to the required precision<sup>9</sup>. In so doing, the students learned a practical use for the mathematical concept of Taylor expansions even though the function is known, so one could easily think that there is no use in approximating it. Both of these examples showcase calculations that only require the basic four arithmetic operations, which computers are built to excel at.

In short, students can learn a lot about mathematics by working with numbers on the computer, not just the other way around. These two-way *connections* between mathematics and computing can be interpreted as a form of *transfer* between the domains of computing and mathematics, which I will examine more closely in Section 3.2. But I will begin my coverage of the theoretical background by looking at how students make sense of mathematics and computing more generally.

---

<sup>8</sup>In practice, memory is limited, and if we are working with 64-bit floating-point numbers, such a sum has to terminate at a certain point. This raises the tricky question of how to test for convergence with floating-point numbers.

<sup>9</sup>These values can also be *found* using appropriate Taylor expansions, and Paper III demonstrates the use of one to find the value of  $\ln 0.75$ .

## Chapter 3

# Theoretical Background

This section provides the theoretical background for the work, and is divided into three subsections:

- Section 3.1: The *Understanding by Design* (UbD) framework
- Section 3.2: Actor-Oriented Transfer (AOT)
- Section 3.3: Sensemaking

Each of the three subsections forms the foundation for one of the three papers. Some of the material in this section also appears in the papers themselves. Here, I will elaborate on that material and also clarify the connections between these three theoretical pillars.

### 3.1 The *Understanding by Design* Framework

I based my tutorial designs on the framework of (Wiggins and McTighe, 2005), which describes a *backwards design* process of three stages:

1. Attaining clarity of the learning goals and defining the understandings that students should come to.
2. Determining what would be acceptable evidence for this understanding having taken place, and design assessments to uncover that evidence.
3. Finally, designing the learning activities by which the students will be able to uncover the desired understandings.

In the Understanding by Design framework, an *understanding* is defined as a specific and useful generalisation that points to transferable big ideas and requires uncovering and insight to grasp, as opposed to mere drill. Using the definition of *transfer* to mean any generalisation students make, without focusing on correctness (Lobato, 2012), we note that Wiggins and McTighe echo this sentiment in their discussion of assessment *validity*: "we typically pay too much attention to *correctness* [in our assessments], and too little attention to the *degree* of understanding" (Wiggins and McTighe, 2005, p. 183). In other words, we often fail to take into account the degree to which performance and understanding are correlated. This is a potential pitfall both in traditional assessments and traditional transfer studies.

According to Wiggins and McTighe, there are six kinds of understanding: being able to (a) *explain* general ideas, (b) *interpret* specific instances of such ideas, (c) *apply* the ideas and knowing when and how to use them, (d) gain

### 3. Theoretical Background

---

distance to the subject matter and see it from different *perspectives*, (e) have *empathy* with ideas that seem odd or foreign at first glance, and (f) have *self-knowledge* so as to know what one knows, what one does not know and how one's learning is progressing.

In the first phase of backwards design, where learning goals are in focus, it is important to prioritise. From least to most important, the curriculum is divided into knowledge that is (a) worth being familiar with, (b) important to know and do, and (c) the big ideas and enduring understandings that everything else hinges on (Wiggins and McTighe, 2005, p. 71). For the latter especially, a set of *essential questions* may be a useful tool for the teaching designer. These are not answerable in finality with a brief sentence but meant to have students ponder them and in so doing uncover the understandings we desire. In short: understandings make use of facts but are not simple facts themselves.

The second design phase focuses on evidence for understanding and assessment, and here it is crucial to distinguish internalised flexible ideas from borrowed expert opinions delivered on cue. This involves supplementing the traditional quiz or test with academic prompts and performance tasks. *Academic prompts*, of which our tutorials are examples, pose questions or problems that require critical thinking, explanations and defence of the answer and methods. *Performance tasks*, on the other hand, ask students to do authentic work that yield tangible products and performances and give students opportunities to personalise the task.

In designing tasks, Wiggins and McTighe propose a set of design prompts called GRASPS. The designer should consider the:

- Goal of the task
- Role of the students
- Audience for their work
- Situation that frames the task
- Product/Performance the task results in, and
- Standards by which the work will be judged.

For us to say that they understand, the students need to provide reasons and support for their choices, in line with the six facets of understanding. It is important that the students' answers are not dependent on blatant cues.

Finally, the third phase of backwards design places the focus on learning activities, of which direct instruction (teaching) is but one example. The optimal designs provide students with engaging and effective tasks. An *engaging* task is recognised as meaningful and intellectually compelling by the learners and presents them with a mystery or challenge they can go hands-on with. *Effective* tasks are ones that help learners become more competent. Their goals of such tasks are clear, the criteria are well known, and the students are given opportunities to self-assess along the way.

Another set of design prompts called *WHERE TO* are suggested as a tool to analyse the learning activities. These ask the designer whether they have made clear to the students:

- Where the unit is headed (and Why),
- Hook (and Hold) their attention, and allow them to
- Experience doing the subject,
- Rethink (and Reflect) along the way,
- Evaluate their strategies,
- Tailor and personalise the task to their own preferences, and
- Organise the activity using a whole-part-whole format.

The final versions of my three tutorials (Sections A to C), make use of these design prompts in the following ways:

- Where/Why: Introductory sections explaining the overall goal of the tutorial, and the reasons for it.
- Hook/Hold: An element of mystery that I found piqued the students' interest:
  - "What is wrong with this program?" (Tutorial 1)
  - "How can we make our own logarithm function?" (Tutorial 2)
  - "What is the fastest numerical integration we can do that is still accurate?" (Tutorial 3)
- Experience: The students get hands-on experience debugging, implementing, and designing programs to solve mathematical problems.
- Rethink/Reflect: I inserted prompts into the tutorials encouraging the students to reflect on what they had done so far.
- Evaluate: Students were given opportunities to self-assess.
- Tailor and personalise: The students were given real choices of how to implement things. We highlighted some decisions they might not be aware of making otherwise.
- Organise: The prompts under "Rethink/Reflect" also allowed us to focus on the big picture whenever the students had worked with details for a significant amount of time.

#### 3.1.1 Black Box Thinking

While not formally a part of the UbD framework, I also found the concept of *black box* thinking<sup>1</sup> useful for my third paper. We borrowed the term from computer science education literature (see for instance du Bolay et al., 1981). Aaron Falbel argued that tools, such as computers, should be transparent and *convivial*, in the sense that the social arrangements people create around their use afford the users of these tools to invest the world with meaning:

In some ways, the design of computers is becoming less convivial. The simplicity (transparency) of the early models has evolved toward complexity (opacity). When home computers started to appear in the mid-1970s, they were often sold in kit form to be assembled at home by computer hobbyists. [...] These early computers were designed to be tinkered with. Not so anymore. [...]

The computer is becoming a veritable black box. And in the process, we are witnessing a reduction in conviviality, the hallmark of which is self-reliance. The message on the label is clear: You cannot understand this machine; you must rely on the experts. [...] You cannot customize it. You cannot “look under the hood” to see how it works. And while it is possible to use a computer without knowing how it works, how to program it, or how to repair it, such use limits our freedom. We must settle for whatever the experts send our way. [...]

Learning to program a computer can increase its capacity for conviviality and can help counteract the current trend toward anticonvivial hardware. Knowledge of a programming language enables computer users to shape the tool to their needs and tastes. This allows for further freedom, choice, and flexibility because it encourages users to determine what computers can be made to do and what they can be used for. (Falbel, 1991)

For most students, asking an electronic device for the logarithm of a number can be characterised as such a black box operation: The number is returned as if by magic, with no reference to the means of its calculation, nor any measures of its accuracy. While students learn to depend on these answers’ correctness<sup>2</sup>, there is little understanding involved beyond figuring out which buttons to press (Gravemeijer et al., 2017; Watters and Watters, 2006).

Importantly, we should be careful not to say that the student is *unable* to understand, but rather that they have not engaged with how the result was

---

<sup>1</sup>Not to be confused with the black boxes used in airplanes and the insight they provide in accident analysis. That concept by the same name, while evidently also useful, ironically does provide the transparency that is missing in our use of the term.

<sup>2</sup>While we all have to depend on black boxes from time to time, the danger is that we do not understand their limitations. Not everyone can be expected to have this understanding, but the experts that we educate should, especially if the black box is used for computations of critical importance.

found as something to be understood. In these cases, it may simply be that the task does not require students to attend to this aspect: all that is asked of them is that they get a correct answer without an explanation of how that answer was derived.

Thus, while black box thinking can be said to represent knowledge in the sense that the students know how to formulate a query of the computer to get an answer, understanding in our context means that they also know how the computer finds the answer, and that they are able to interpret and connect it to other forms of knowledge as well. Black box thinking, then, is not what we would consider to be understanding, but it is nonetheless particularly relevant for contexts that involve computing, and we saw an interesting example of this in one of our interviews (see Paper III).

## 3.2 Actor-Oriented Transfer

As I mentioned in the introduction, transfer of knowledge can be regarded as one of the foundations for the existence of schools and universities. However, there is growing evidence that rather than being a product of institutional practices, transfer is a process shaped first and foremost by the *learners* (Billett, 2013). This suggests that the attending to the student perspective is crucial when attempting to study transfer of learning.

The *actor-oriented transfer* (AOT) perspective (Lobato, 2003, 2012; Lobato and Siebert, 2002) is a theoretical framework that takes a student-focused view of transfer. Here, "transfer is defined as the generalization of learning, which can also be understood as the influence of a learner's prior activities on her activity in novel situations" (Lobato, 2012, p. 233). This applies even when students make unexpected connections that may or may not result in incorrect performance, and stands in contrast with more traditional views of transfer, in which students are only considered to be transferring knowledge if they correctly solve transfer tasks that are predetermined by researchers/observers (Lobato, 2008).

In AOT, taking the actors' point of view<sup>3</sup> affords making explicit the elements of mathematical understanding that can remain implicit in traditional transfer studies that employ an observer's point of view. For this reason, AOT is regarded as a useful tool in iterative design-based research studies that seek to improve instruction and the ways in which students generalize (Lobato, 2012).

AOT emerged out of design-based research where a traditional transfer failed to capture the students' generalisation and the similarities they noticed. The usefulness of AOT in design-based research comes from its power to inform design decisions and identify what is salient for students. AOT and design research both focus on learning processes, learner-centric classrooms and the recognition that learning is social as well as cognitive (Lobato, 2003).

As we have seen, an important facet of the AOT framework is its focus on similarities and dissimilarities as seen from the student's point of view. These

---

<sup>3</sup>In our context, the actors are the students that I interviewed.

### 3. Theoretical Background

---

allow actors to make *connections* between the activity they find themselves in and some previous activity that they regard as similar and relevant. (Lobato, 2012) quotes (Hohensee, 2011) in saying that the features that the students notice are

conceptually connected to the ways in which students transfer their learning experiences. [It] is unlikely for a teacher to simply say ‘Look here!’ and her students will notice what she targets. Instead, there is a system of elements (discourse practices, mathematical tasks, and the nature of mathematical activity) that work together to bring forth the noticing of particular mathematical features in classrooms. (pp. 242-243)

This is not to say that context is irrelevant in the AOT perspective, nor that it only serves as a barrier to transfer which learners can overcome by abstraction. Combining AOT with a *knowledge-in-pieces*<sup>4</sup> perspective (diSessa, 1993; Wagner, 2010) found that the formation of abstract representations is not the only way in which transfer occurs. It may also be supported through incremental growth and organisation of pieces of knowledge that are highly sensitive to context.

I investigate students working with knowledge from the two domains of computing and mathematics in the same setting it has been taught, which does not require proof of transfer to a *different* context. That is to say, the actor-oriented transfer that I investigate occurs between domains, but the context of my tutorials is not fundamentally different from the one that our students are familiar with from their classes.

I use the term *affordances* to describe what benefit students derived from making connections across domains. The term has often been used in the mathematics education research literature, most notably in the context of technology. The term describes a relationship between an actor and an object that is expressed by some activity: an illustrative example by (Gibson, 1979) is that water affords drinking and drowning to human beings, while it affords breathing to fish (Brown et al., 2004). In our context, when I discuss affordances and limitations, I am concerned with the activities that are helped or hindered, respectively, as a result of interactions between humans and computers.

A recent example that uses the AOT framework in the context of computing comes from (Lockwood and De Chenne, 2020), which explores how students use Python to list and count the outcomes of combinatorial counting problems<sup>5</sup>. One way to achieve this is using conditional statements to eliminate outcomes that do not respect restrictions on ordering, repetition, or both<sup>6</sup>. Lockwood

---

<sup>4</sup>Imagine knowledge as a relational network where the nodes - the pieces - are elements that can be linked together in different ways to constitute knowledge. Concepts can be linked together in this way, but may also have internal structure of finer-grained knowledge pieces that are nested in a similar way. DiSessa likens the pieces that make up a concept to the ingredients of a cake, and knowing their relations to having a full recipe (diSessa, 2014).

<sup>5</sup>Combinations (unordered) and permutations (ordered), with and without replacement.

<sup>6</sup>For any two elements  $a$  and  $b$  that belong to the same set of numbers, we can insist that they be ordered ( $a \leq b$ ), unique ( $a \neq b$ ) or both ( $a < b$ ).



and De Chenne found that this approach focused students' attention on the outcomes they were counting, which again reinforced the conceptual differences between different types of counting problems.

I consider the AOT and UbD frameworks to be complementary. Where AOT takes the point of view of the learner, UbD takes the teacher or teaching designer point of view. UbD recognises that students making connections is important and goes about exploring how one might initiate or support this. Therefore, using AOT to describe these connections and identify the features of the learning environment that supported these is important for the designer as well. Furthermore, describing the affordances of such connections may help motivate and explain design choices.

### 3.3 Sensemaking

The challenges facing modern humans and scientists are complex, ranging from global warming (Ryghaug et al., 2011) to pandemics (Weinreich et al., 2021) to figuring out how the evolution of the universe is affected by gravity and the properties of elementary particles (Sand, 2016). Most of these complexities demand building coherent theories, comprehending *why* something is, and when to use and not use what we know. This goes well beyond simply knowing or recalling the *facts* (Wiggins and McTighe, 2005). In other words, we require our students to make sense of things.

As computers become more powerful and interconnected, the wealth of information available to any of us can be overwhelming. Modern education requires educators to equip students with the skill of critical thinking, to separate facts from opinions, and connect facts in a consistent way with the power to provide explanations of these complex phenomena. An important aspect of teaching critical thinking is therefore to encourage and model the process of *sensemaking* (Maloney, 2015; McPeck, 2016).

In the education research communities, sensemaking has taken on many different meanings. A recent example from mathematics education research can be found in (Biccard, 2018). The definition of sensemaking I use in this thesis is from physics education research: "A dynamic process of building or revising an explanation in order to [...] resolve a gap or inconsistency in one's understanding" (Odden and Russ, 2018), pp. 5-6.

While there have been numerous other attempts to define what sensemaking is, I chose this one since it unifies several aspects of sensemaking that others have highlighted: sensemaking as an epistemological frame, a cognitive process, and a discourse practice, all three of which are rooted in the science education literature.

Sensemaking as an *epistemological frame* concerns how students approach learning-based activities and what they think is going on with those activities. Sensemaking then becomes such a frame when students interpret their task as constructing an explanation for something not understood or figuring something

### 3. Theoretical Background

---

out. The foundation for this aspect comes from the *resources-based* perspective, see for instance (Hammer et al., 2005).

Sensemaking as a *cognitive process* is built on the theory of *knowledge integration* (see among others Chiu and Linn, 2011), which is a process where students articulate, consider, compare and possibly integrate different mental models of a phenomenon. Some authors term this process as generating *self-explanations* (Kapon, 2017).

Sensemaking as a *discourse practice* involves *argumentation* - constructing and defending arguments. This is a different goal than persuasion. It can be broken down into two sub-processes: *construction* and *critique* of arguments. These can be done in a group or within the mind of one person (Ford, 2012).

In (Odden and Russ, 2018), the process of sensemaking involves the following steps, illustrated in Figure 3.1:

1. becoming aware that there is a gap or contradiction in one's knowledge,
2. proposing ideas and attempting to connect them to existing knowledge or other ideas, and
3. evaluating that these ideas are consistent and do not lead to additional contradictions.

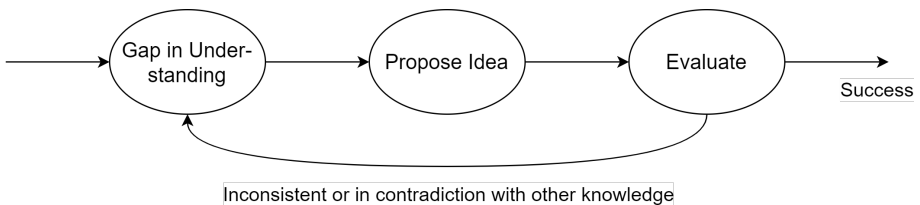


Figure 3.1: The sensemaking process, according to ((Odden and Russ, 2018)).

This process can be iterative: if an idea proves to be troublesome, one must try a revised suggestion that ensures the theory is not internally inconsistent or contradicting the external facts.

Sensemaking is far from the only theoretical framework attempting to describe learning at this level of granularity. Another example is *conceptual change*, which is inspired by paradigm shifts in research communities and aims to illuminate implications for education. It is assumed that the scientist and the student both have a current concept and see a rational part to learning (student) or adopting (scientist) a new one. Strike and Posner outline four conditions for conceptual change to occur: (a) dissatisfaction with existing conceptions, (b) new conceptions being intelligible, (c) new conceptions appearing plausible, and (d) new conceptions being able to open up new areas of inquiry (Strike and Posner, 1982).

The first of these four points strongly resembles the first step of Odden and Rush, under the assumption that one is not satisfied with one's knowledge

being incomplete. Similarly, a new concept being intelligible and appearing plausible allows it to be proposed to bridge the gap. In contrast to Strike and Posner sensemaking as Odden and Russ define it does not require that new areas of inquiry being opened up: it may happen or it may not. In that sense, sensemaking is more concerned with the immediate process and less so with its after-effects. One way to interpret this difference is to propose that sensemaking is a process that may lead to conceptual change if successful. On the other hand, Strike and Posner's version of conceptual change leaves implicit the evaluation that sensemaking emphasises in its third step, and the ideas that are rejected in the process. I thus view these perspectives as complementary.

Another perspective on conceptual change is that of diSessa, who approaches the problem from the knowledge-in-pieces perspective (Section 3.2). DiSessa argues that if concepts are components of larger-scaled systems, the systemic constraints can constrain individual concepts to the effect that changing a concept becomes difficult. DiSessa argues that "a nearly unique property of [knowledge-in-pieces] in the field of conceptual change is that it sees 'naïve' students as full of ideas, many of which can or even must be re-used in developing scientific understanding" (diSessa, 2014, p. 12), though these ideas are usually less coherent than those of experts.

In that sense, diSessa's perspective is more focused on the here-and-now of learning than Strike and Posner. Compared to sensemaking as defined by Odden and Russ, the most striking difference is grain size - knowledge-in-pieces conceptual change may operate at a finer grain size than sensemaking. Both perspectives can operate on a short time scale, however, as demonstrated in (diSessa, 2017). The event described by diSessa in this paper resembles a sensemaking process: a student's intuitive ideas about thermal equilibrium is challenged by experiment, whereupon the student suggested an explanation<sup>7</sup> for the observed behaviour. As with Strike and Posner, diSessa's analysis seems to leave implicit the evaluation of these ideas and focus more on their construction.

It is not obvious that sensemaking is the superior theory to analyse the events in our data. Conceptual choice might very well have produced valuable insights and both theories are appropriate to use on short time scales to investigate learning that is more than rote acquisition of simple facts. The reasons we chose sensemaking were its larger gain size, deemed more suitable for a first exploratory study and its focus on evaluating ideas for consistency, a process which features heavily in our data, as it turned out.

Sensemaking is also related to the UbD framework. The most prominent links between the two theories are:

- When considering what students should make sense *of*, the UbD framework provides clarity on the learning goals (in the first phase of backward design).
- When considering how to design learning activities (in the third phase), the UbD framework describes how to design for sensemaking to take place.

---

<sup>7</sup>The further liquids are from equilibrium, the more they "freak out" and the harder they work to regain equilibrium (diSessa, 2017).

### 3. Theoretical Background

---

- The UbD framework describes what understanding is, and how to identify learning goals and activities that promote this understanding from the teaching designer's point of view. It does not describe in detail the process of coming to an understanding from the students' point of view. Sensemaking provides this very piece of the puzzle.
- The role of essential questions in the UbD framework (see Section 3.1) as a strategy to identify what is important to understand and to promote students engaging in the big ideas is mirrored in sensemaking. These questions require uncovering and are not answerable with just facts, and (Odden and Russ, 2019) posit that such questions can initiate and sustain a sensemaking process.

Sensemaking is also related to Actor-Oriented Transfer. Sensemaking describes the *whole* of the process of constructing new understandings, whereas AOT focuses on the *parts*: the individual *connections*, such as similarity or dissimilarity, that make up the sensemaking process. Indeed, with AOT's connections we are approaching a grain size comparable to knowledge-in-pieces conceptual change<sup>8</sup>.

To illustrate the overlap between sensemaking and AOT with a simple example, the following occurred in the writing of Section 2.2:

I know that the exponential standard form of a binary number is a number between 0.5 and 1 multiplied with a power of two, and this is the format returned by Python's `frexp()` function, which returns the mantissa and exponent. I also know that the mantissa and exponent of a floating-point number are stored in memory as binary integers. Here is a *dissimilarity* – the author makes the connection that the mantissa cannot both be an integer and a number between 0.5 and 1. This connection and what follows from it constitutes stage 1 of the sensemaking process.

I propose that the mantissa *is* in fact stored as an integer, and that the computer then *interprets* that integer as the numerator of a fraction (stage 2 of sensemaking).

In evaluating this idea for coherency and looking for contradictions, I find a *similarity* – if that fraction is a number between 0.5 and 1, the two representations could be one and the same. This connection contributes to stage 3 of sensemaking.

This process might well be cyclic, requiring more three-stage iterations to resolve fully. For instance, if "bbb..." denotes the individual bits (b) in the mantissa, the standard for floating-point arithmetic specifies that the mantissa is interpreted as "1.bbb...", which is clearly not a number between 0.5 and 1, but between 1 and 2. This differs notably from what the `frexp()` function returns. A subsequent sensemaking iteration, however, could have us discover that these numbers differ only by a factor of 2, which could be pulled into the exponent with little difficulty.

We will see a similar process at work in Paper I where we encounter a student trying to make sense of an apparent contradiction concerning the relationship

---

<sup>8</sup>In fact, we recognise the data in diSessa, 2017 as something that could very well be analysed in terms of connections and similarities.

between mathematical accuracy and realism. In that paper, we did not consider individual connections in the way that I did here, but I will demonstrate that this is possible in Section 7.1.



## Chapter 4

# Methodology

With the theoretical landscape of Figure 1.7 thus established, we are nearly ready for the story of my path through this landscape, with the three papers of Figure 1.8 as important milestones along the way. But first, we need to summarise the methodology common to the three papers.

### 4.1 Iterative design process

For all three papers, design of the learning activities was done iteratively:

- I proposed design ideas for faculty teaching the course whose learning goals the designs were based on.
- After incorporating their feedback, the initial design was tested by interviewing students as described in Section 4.2.
- Between individual interviews, minor adjustments and clarifications were done as necessary based on the interview data. Typically, all interviews in one such cycle happened during the same week, so the students were roughly in the same place in the course schedule. This allowed at least rough comparisons between interviews in the same cycle.
- Larger changes were done in between cycles of interviews. These reworkings could take from several months up to one year. In these design phases, the lessons from the entire set of interviews using the previous version were incorporated, with additional input from teaching faculty and the research group.
- Successive interview cycles brought in additional data, which were used to finalise the design.

The three tutorials for Paper II and III were all designed in this way. The versions I include in the appendix are the final versions, which may differ in important ways from those used for the papers. The lessons learned that I describe in the papers have been incorporated in these final versions. All versions of the tutorials were, with a few exceptions, also used in class alongside the interview cycles.

For Paper I, I found the interview task too short to deserve to be labelled a full tutorial. Unlike the tutorials, this task was never used in class, and only served as a pilot experiment to get research data. For that reason, I have opted to not include it in full, for the following reasons:

## 4. Methodology

---

- Paper I describes the task in sufficient detail for researchers to reproduce and for educators to take inspiration from, should they wish.
- This task was very exploratory, and I do not deem it to be "classroom ready". Including it in the appendix might give the impression that it is as finished as the tutorials that I did test in class, with observations, which is not the case.

Finally, I note that the tutorials were designed using the backward design process described in Section 3.1.

### 4.2 The Interviews

I recruited students in two ways: (a) asking them to volunteer using an online form during one of the first lectures of the semester, and (b) recruiting groups of students during in-class observations. I conducted all the interviews in person using both video and screen recording on a computer where a familiar Python programming environment had been set up for the students to use<sup>1</sup>. They also had access to a whiteboard, which the video camera captured.

After these pilot interviews, we changed our focus from individual think-aloud interviews (as in the case of Sophia in Paper I), to interviews with groups of students. This we did for the following reasons:

- The think-aloud interviews, while illuminating, are not realistic in terms of how students work with exercises in class. Instead, the interviewer tended to take on the role of both Teaching Assistant (TA) and fellow student to discuss with. Therefore, we found that interviewing students in groups would provide more realistic data and make our conclusions easier to apply to teaching in the classroom.
- Having the interviewer play a more withdrawn role, only intervening when necessary or requested by the students, would similarly add to the realism of the learning activity and usefulness of the data. For generalisability and reproducibility, we would prefer students discussing the work with each other, not with an expert.
- Inviting three students proved to be optimal, as sometimes individual students had to cancel. Thus, I could interview the remaining pair of students instead of cancelling the entire interview or switching to think-aloud because I only had one student left.
- With two to three students, at least in theory the cognitive load would be spread across more people, which would free up students' cognitive resources as some of the complexities involved would be handled by the other students (Costley, 2021).

---

<sup>1</sup>JupyterHub for the BIOS1100 students of Paper I, a code editor (Atom) and a terminal window (Anaconda) for the MAT-INF1100 students of Paper II and III.



- Given our research interest in connections especially, we found it more likely that more interesting connection patterns would surface between groups of students than with only one student, as they could build on each other's connections.

In these group interviews, the students were told to work together as they would in a normal group session in the course, using the interviewer as a TA where necessary. The authors collectively decided on follow-up questions to gather feedback for the next design cycle and record the students' previous experience with programming, which might differ substantially and would make us alert to cases where the student's actions might be influenced more by previous experience than by the tutorials themselves.

These follow-up questions were generally (with the exception of Paper I) saved for the end of the interviews. While this tends to affect validity, as students then have to recall their experience from earlier in the interview, it ensures that their thought processes during the interviews themselves are affected as little as possible by the interviewer (van Someren et al., 1994). General follow-up questions pertaining to the students' background and their thoughts on using computing and mathematics together were agreed on by the research group ahead of time. Other follow-up questions would ask students to elaborate on certain choices or actions during the interviews: these were noted down during the interview and asked at the end.

## 4.3 Analysis

In this section, I describe my analysis in broad terms: more detail will be provided in each respective paper. Common to all three papers is that they are exploratory case studies, and no pre-existing code book was available for use in the analysis. We therefore performed a *thematic analysis* of the data, following roughly the six-phase process outlined in (Nowell et al., 2017).

### 4.3.1 Phase 1: Familiarising Yourself With Your Data

As a first step, the interview audio was transcribed at the end of each semester. I reviewed the transcripts to divide them into short segments with brief descriptions of each. Segments were flagged for further review if they involved both mathematics and computing, and the students' work was focused on understanding something, as opposed to performing a skill or recalling simple facts.

This selection process produced several episodes (usually consisting of several consecutive segments) where students either (a) were engaged in some activity that contained elements of both computing and mathematics, (b) made a transition between computing and mathematics, or (c), both. I then translated the transcripts from Norwegian into English and enhanced them with evidence from other sources, such as screen and video captures, photos of the whiteboards and collected worksheets. These enhanced transcripts were shared with the rest

## 4. Methodology

---

of the research group (the co-authors) and formed the basis of the analysis from that point onward.

### 4.3.2 Phase 2: Generating Initial Codes

Initially, I divided each episode into short segments and coded each segment. We then looked at the bigger picture, focusing on the patterns that were apparent from the students' point of view. Based on this, I formulated claims relevant to the particular paper, supported by evidence from the transcripts, in analytical memos. The co-authors of each paper reviewed the claims and their justifications for the sake of validation and helped refine the claims in several steps. Examples of these early codes showing our interest in the integration between mathematics and computing were: pure coding, pure math, integrated<sup>2</sup>.

### 4.3.3 Phase 3: Searching for Themes

We assigned labels to the episodes, identifying and highlighting key actions that we took as justifications for this coding. Examples of these include verbal utterances from the students, things they wrote in the code editor or the whiteboard, and other actions. In the process I kept detailed notes of each version of these descriptive labels and triangulated the labels with the co-authors. Examples of early themes or labels were: "mapping code to mathematics", "mapping mathematics to code" and "explicitly telling the code what to do".

### 4.3.4 Phase 4: Reviewing Themes

We created new enhanced transcripts with a separate column for coding and key actions to get a clearer picture of the codes as applied to raw data, including data we had not previously analysed in detail. In the process, we also linked the key actions to theory. For example, for my second paper, we noticed that a lot of our key actions were students making connections or noticing similarities.

### 4.3.5 Phase 5: Defining and Naming Themes

We summarised what each of the selected episodes demonstrated and refined our coding of these based on this, before going back into the data and re-coded them using the updated labels. In the process, we collectively decided on names for the labels in several steps. This resulted in the label names and definitions presented in Table II.1 of Paper II.

### 4.3.6 Phase 6: Writing Up the Report

Finally, we started writing the papers, initially focusing on describing the themes and the process leading to their development. We referred back to the theoretical framework to justify our choice of themes, and attempted to articulate what

---

<sup>2</sup>This example, and the examples in the following subsections are all from Paper II

each theme meant and revealed about the topic. In our second paper this appeared in the form of patterns of connections and what each pattern afforded the students (a step up in grain size). We also sent finished drafts of each paper to all respondents to establish the fit between their views and our representation of them.

For further details as to how each individual theoretical framework was applied in the course of the analysis, I refer to the papers themselves (Paper I, II and III).



## Chapter 5

# The Path to Paper I

It is now time to turn our attention to the papers themselves. In order to contextualise the papers, this section gets us started along the chronological path through the theoretical landscape outlined in Figure 1.7 and Figure 1.8. What follows in this section through to Section 7, is the story of what I did and what I learned from it. The papers themselves appear as separate chapters where they belong chronologically. This section represents the beginning of that story and focuses on the early work in 2017 and 2018 that culminated with Paper I.

The first paper sprung out of classroom observation in the course *BIOS1100: Introduction to computational models for Biosciences* (Nederbragt, n.d.). This compulsory course for first-year bio-science students integrates both mathematics and computer programming to help students solve relevant biological problems, such as bacterial growth, DNA sequence analysis, and disease spreading through a population.

I would like to stress that this was the very first semester the course was taught, and as such it represented the first of three phases that I have observed in the evolution of the course since then:

1. Main focus on students learning programming – presence of mathematics mostly implicitly.
2. Explicit focus on mathematics but taught alongside programming with little integration of the two domains (thus not following the approach outlined in Section 2.1).
3. A movement toward integrating mathematics and computing.

My observations, in addition to help motivate and lay the groundwork for my research, also led to some changes in the course after the first semester. These came in addition to the great effort the course teachers put into improving the course over the next years.

In the first semester, when the course was still in phase 1, I was left with an impression that the students possessed a set of computational and mathematical skills, but they had a hard time making sense of how to use them effectively. To probe this further and make use of sensemaking as described in Section 3.3, I recruited volunteers from the class and interviewed them as they worked on a task that was designed to highlight the ways in which they made sense of what they were doing. This would specifically allow further insight into how these students worked with loops and variables.

During these pilot interviews, we noticed something interesting. The original task was designed to shed light on how the students made sense of the program

## 5. The Path to Paper I

---

they were writing. The task, written in the *JupyterHub* environment with which the students were familiar from class, can be summarised as:

Given an initial population of 10 rabbits that increases by 10% every month, calculate by hand the number of rabbits after 1 and 2 months. Perform the same calculation in Python and print the results. Make a loop that calculates the number of rabbits for the first 10 years and plot the results.

The interesting thing that surfaced in this interview data, was that the students very quickly got fractions of rabbits:

*Month 0:* 10 rabbits  
*Month 1:*  $10 \cdot 1.1 = 11$  rabbits  
*Month 2:*  $11 \cdot 1.1 = 12.1$  rabbits

This brings into play a trade-off or conflict between accuracy and realism: Mathematically, 12.1 rabbits more accurately represents 110% of 11 if we abstract away the context, but even if you could have an extra 10% of a rabbit, the idea that this fraction of an animal could contribute to further population growth is absurd. As one of the students said in the follow-up session:

**Interviewer:** In your calculation – both on paper and in Python – you got 12.1 rabbits after 2 months. How realistic do you feel this answer is?

**Student:** Very unrealistic because you can't have 0.1 rabbit [laughs].

**Interviewer:** Do you remember what you were thinking when you chose not to round it? I remember that you rounded to 3 decimals [in your later calculations], but not to integers. And what do you think the choice of rounding or not will mean for the correctness of your answer after 120 months?

**Student:** I chose 3 decimals because I thought it gave a prettier answer, which was easier to relate to. I didn't round the final answer because I was unsure how to get the `round()` function into the `print()`. Whether that affects the answer after 120 months in my opinion depends on whether it's the rounded number or the number with decimals that is used for further calculations. If the `round()` is only in the `print()`, then it shouldn't matter in my opinion. [...] If the answer should be as realistic and accurate as possible, I would calculate with all the decimals – which is what I did – and then round the answers in the end.

Note that this choice still implies that mathematical correctness takes precedence over realism: in this interpretation, we still have fractions of rabbits contributing to the population growth, even though the answers are rounded after all the calculations are done. If the student had favoured realism, only

---

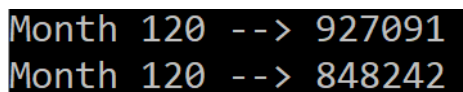
whole rabbits (or even pairs of whole rabbits) could have contributed to the growth of the population, and as the student observes, this would have led to a very different answer, as seen in Code Sample 5.1 and Figure 5.1:

```
r_a = 10
r_b = 10

for i in range(1, 121):
    r_a = r_a * 1.1
    r_b = round(r_b * 1.1)

print("Month", i, "-->", round(r_a))
print("Month", i, "-->", r_b)
```

Code Sample 5.1: Python code illustrating the difference between two approaches to rounding: during and after the calculation (the author's work, not the student's).



```
Month 120 --> 927091
Month 120 --> 848242
```

Figure 5.1: Output of the code in Code Sample 5.1.

There is, however, a way to reconcile these two approaches, as we will see in the next section. There, I will describe one of the interviews I conducted based on an altered version of the task, designed to bring this very issue to the forefront. For the last batch of these pilot interviews in the spring semester of 2018, I re-worded the exercise to involve decaying radioactive nuclei, for three reasons:

- these students were taking an introductory physics class that semester, so the task seemed relevant to them in that sense,
- if the students could not make sense of the task with this framing, I could re-frame the task for them in terms of rabbits and see if bringing it into a context that was more familiar for them changed anything, and
- decay would highlight the issue of rounding (see below).

For radioactive nuclei, exponential decay was a more natural choice than the exponential growth typically associated with rabbit populations. Here, the issue of whether to round during or after calculation that I discussed in Section 4 would be even more pronounced: if we round the number of nuclei *during* calculation, we can never go below 4 nuclei. If we start with 1000 nuclei and 10% of them

## 5. The Path to Paper I

---

decay every month, we eventually get<sup>1</sup>:

$$4 \cdot 0.9 = 3.6 \approx 4$$

This is simply not how the world works on small scales. Any radioactive nucleus, even if left to itself, will eventually decay spontaneously. Its *half-life* (the time it takes 50% of nuclei to decay), is simply a statistical expectation value that is only meaningful when we have a large population of nuclei. The time it takes an individual nucleus to decay is governed by quantum mechanics and is inherently random. At this point, stating that "10% of nuclei decay every month" becomes physically absurd<sup>2</sup>, while mathematically, there is nothing wrong with taking this statement at face value and insisting on calculational accuracy. My task was therefore designed to produce situations with very few nuclei, so I could investigate the students' processes of making sense of this apparent contradiction.

The task I gave the students used this very set-up and I asked the students how many months it would take for there to be no radioactive nuclei left. Some of the students were completely fine with having fractions of nuclei until I re-framed the situation as an exponentially declining rabbit population. At that point, they realised that fractions of rabbits did not make sense and made the logical connection that fractions or nuclei (or "atoms" as the task called them, since we would not expect the nuclei to be bare of electrons) did not make sense either.

One student, whom I gave the pseudonym Sophia, went much further than this, however. My first paper is devoted in its entirety to the interview with Sophia, which I present in the next section.

---

<sup>1</sup>Mathematically, one would expect this calculation to stop at 5. In Python 3, however, floating-point numbers equidistant from two powers of 10 (that is, ending with the digit 5) are rounded toward the nearest *even* number. This so-called *banker's rounding* ensures that there is no accumulated bias toward higher numbers if we add together many such numbers, as the following example shows:

Normal rounding:  $8 = 0.5 + 1.5 + 2.5 + 3.5 \approx 1 + 2 + 3 + 4 = 10$

Banker's rounding:  $8 = 0.5 + 1.5 + 2.5 + 3.5 \approx 0 + 2 + 2 + 4 = 8$

See also <https://docs.python.org/3/library/functions.html#{#}round>.

<sup>2</sup>We do not expect students to know this in an introductory physics course. This made the problem an interesting one, as it can appear to be self-contradictory until they make sense of it. The isomorphism with rabbits was also helpful: if we stated the problem such that 10 percent of a rabbit population dies every month, students were less likely to interpret this as the death of partial rabbits.



## Paper I

# How Computation Can Facilitate Sensemaking About Physics: A Case Study

**Odd Petter Sand, Tor Ole B. Odden, Christine Lindstrøm, Marcos D. Caballero**

Earlier version published in the *2018 Physics Education Research Conference Proceedings* <https://www.compadre.org/per/items/detail.cfm?ID=14850> (2018)

### Abstract

We present a case study featuring a first-year bio-science university student using computation to solve a radioactive decay problem and interpret the results. In a semi-structured cognitive interview setting, we build on previous work on sensemaking by studying the process in a computational science context. We observe the student using computation as an entry point into the sensemaking process and then making several attempts to resolve the perceived inconsistency, drawing on knowledge from several domains. The key to making sense of the model for this student proves to be thinking about how to implement a better model computationally. We demonstrate that integrating computation in physics activities may provide students with opportunities to engage in sensemaking and critical thinking and discuss some implications for instruction.

### Contents

I.1	Introduction . . . . .	45
I.2	Analytical Framework . . . . .	46
I.3	Methods . . . . .	46
I.4	Computational Sensemaking Case . . . . .	48
I.5	Discussion and Conclusions . . . . .	51
	References . . . . .	53

### I.1 Introduction

It is a well-known conundrum that students can progress through introductory physics courses, sometimes with good grades, and still lack understanding of the

## I. How Computation Can Facilitate Sensemaking About Physics: A Case Study

underlying principles, relations, and concepts. A dreaded, but common scenario is students employing "plug and chug" strategies to manipulate mathematical formulae without engaging with the underlying physical principles. With this in mind, getting students to engage in sensemaking is crucial for achieving learning goals in critical thinking and understanding the physics itself Maloney, 2015.

Computation is important for students of physics to learn because it reflects current practices in the field, teaches important skills for research and other careers, and allows students to solve a greater number of more realistic problems "AAPT Recommendations for Computational Physics in the Undergraduate Physics Curriculum", 2016. As a consequence, research-based efforts to sensibly integrate computation into the physics curriculum are well underway Caballero et al., 2012. Therefore, we want to study to what extent computation provides a potential for students engaging in sensemaking, and under what conditions that potential may be fully realised.

We present evidence for sensemaking in the case of Sophia, a bio-science student who is interviewed while solving a physics problem on radioactive decay. Sophia uses both computational and non-computational arguments to make sense of the model she is working with. We claim that because Sophia can easily modify her program and compare the corresponding outputs, sensemaking is facilitated. We justify this claim by presenting evidence for how computation was helpful in Sophia's sensemaking process. Finally, we discuss implications for teaching and future research.

### **I.2 Analytical Framework**

The analytical framework for this study is founded on the following definition of sensemaking from Odden and Russ, 2018, pp. 5-6: "A dynamic process of building or revising an explanation in order to [...] resolve a gap or inconsistency in one's understanding." While there have been numerous other attempts to define what sensemaking is, we chose this one since it unifies several aspects of sensemaking that others have highlighted: sensemaking as an epistemological frame, a cognitive process, and a discourse practice, all of which are relevant to this project.

The process of sensemaking involves (a) realising that there is a gap or contradiction in one's knowledge, (b) iteratively proposing ideas and attempting to connect them to existing knowledge or other ideas, and (c) evaluating that these ideas are consistent and do not lead to additional contradictions Odden and Russ, 2018. In this paper, we will use this definition to study how computational activities may provide opportunities for sensemaking in interdisciplinary science problems.

### **I.3 Methods**

The case comes from a pilot study conducted with first-year bio-science students at a large research-intensive university in Norway. These students learned

computation integrated with biology in the previous semester and were following a physics course in the semester when this study took place. The physics course had not yet covered radioactive decay by the time we interviewed the students. We targeted students with a wide range of self-reported programming expertise who were also comfortable thinking aloud.

Subsequently, we performed a series of semi-structured cognitive interviews in Norwegian where students worked on the task alone. The interviews borrowed heavily from think-aloud protocols (van Someren et al., 1994), but students could ask for help with syntax should they need it, provided they were able to articulate what they wanted the code we gave them to do. Otherwise, they were instructed to articulate as much of their thinking as possible. The interviewer would occasionally ask the students to elaborate on their thinking.

Follow-up questions on students' reasoning were asked by the interviewer on various occasions, interspersed throughout the think-aloud segments. While this tends to change the students' thought processes, so they generally do somewhat better, protocols obtained in this way tend to be more valid than the ones were students recall their reasoning after the fact (van Someren et al., 1994).

We gave the interviewees a toy model starting off with 1000 radioactive nuclei and told them that 10% of the remaining nuclei would decay every month. The students first calculated the remaining number of nuclei for the first two months (where the answers were still integers) by hand. We then had them reproduce these answers by writing a Python program in Jupyter Notebook, the familiar programming environment they used throughout the previous semester. Finally, they were asked to extend the calculations to 60 and 100 months and (if time allowed) plot the results.

This task was specifically designed to allow students to discover a perceived trade-off between accuracy and realism that would require sensemaking to resolve. After a while, you need several decimal points to mathematically describe 10% of what remains, yet when counting nuclei, in general one expects the numbers to be integers. While the toy model we provided may be approximately correct for a large number of nuclei, at lower amounts one would have to interpret the output as an average across many identically prepared experiments for the numbers to make sense.

All the students interviewed ( $N=5$ ) at some point considered rounding the answers to the closest integer to avoid working with fractions of nuclei, although some did this only in response to follow-up questions from the interviewer. Every student also expressed some amount of concern about the mathematical accuracy of their results when rounding the numbers in this way. Two of the interviewees made some progress toward resolving this contradiction by interpreting the un-rounded numbers as an average, one of which was Sophia.

The typical length of an interview was about one hour. All interviews were recorded on audio and video, both of the student and the computer screen. Subsequently, the transcripts were translated from Norwegian into English. We analysed the transcripts using the definition in Odden and Russ, 2018 and looked for the following: The student (a) realising she cannot fully explain the physical phenomenon she is modelling or aspects of the model itself, (b)

## I. How Computation Can Facilitate Sensemaking About Physics: A Case Study

proposing explanations and trying to connect them to scientific or everyday knowledge and (c) evaluating these explanations to ensure consistency.

We then looked at what the student was doing with computation inside and outside of these sensemaking episodes, and asked the following questions: What happens in this computational context when the student engages in sensemaking? Is the computational aspect of the task a help or hindrance to this process?

The case we present illustrates how sensemaking may happen in a computational context. While not the most typical case for this group of students, Sophia's interview was chosen for analysis because her sensemaking was rather explicit in the transcript. Additionally, she ended up using language that was clearly computational to make a profound argument about how to model the physical phenomenon and interpret the results.

### I.4 Computational Sensemaking Case

"Sophia" (pseudonym) is a Norwegian student in her mid-20s, a few years older than most students taking first-year university courses. She describes her experience with programming as one of a fair degree of mastery in most cases. Compared to the average student in the programming course for bio-science students, she comes across as more confident and relaxed than most when working with computer code.

We begin our analysis at the point where Sophia has set up her program to calculate the number of remaining nuclei for the first three months: 1000, 900.0 and 810.0, respectively.

**Sophia [14:35]** *There. Now it's right. [But] now I might want to round these [indicates 900.0 and 810.0] to get... well, just whole numbers.*

In reaction to her program's output, she implements this rounding to the closest integer and checks that it works. Note that she adds rounding when printing the output from the program to the screen, but not in the actual calculations.

**Interviewer [15:05]** *Could you tell me a little more about why you'd round them?*

**Sophia** *Because these are atoms, and you sort of can't have half... or I don't know... it seems a little unnecessary to include, like, 810.0 atoms, in a way.*

We interpret "*you sort of can't have half...*" as that you cannot have a fraction of a nucleus and still call it a nucleus of that particular element, which is a point Sophia returns to later on.

At this point, we have reached the starting point of the sensemaking process. We divide it into three separate segments that correspond to the three ideas Sophia proposes to make physical sense of the numbers given to her by her program.

### I.4.1 Sensemaking segment I

She moves on to the next part of the task, modifying her program to repeat calculations all the way up to 60 months. She inspects the output and indicates the last ten months in the sequence, with 3, 3, 3, 2, 2, 2, 2, 2, 1, and 1 nucleus, respectively.

**Sophia [16:30]** *This looks a little strange... Because here there are no decimals. So... here I'd include the decimals because, like... you can't take 10 percent of... or, I get that you get, like, the same number several months in a row. [indicates the earlier sequence 6, 6, 5, 5] Because 10 percent of 6 is still above 5, like. I'm going to include the decimals.*

While cutting the decimals for large numbers seems fine to her, Sophia realises that for smaller numbers there is something she needs to find an explanation for: The number of nuclei remaining constant for several time steps and then changing considerably more than 10% rather abruptly. Importantly, the sensemaking process starts as a reaction to the computational output.

Using computing also allows her to implement and test this change, which she immediately does. Yet, the argument Sophia makes here is purely mathematical. She talks about numbers in a sequence, decimals and percentages, but this discussion stands on its own removed from the physics and computational contexts it occurred in.

### I.4.2 Sensemaking segment II

After resolving some bugs (one syntax error and a few logical errors), Sophia sees the un-rounded numbers for all 60 months. After verifying that they seem to be the correct numbers mathematically, she is told that she is free to move on to the next task. Still, she hesitates.

**Sophia [20:18]** *Umm, yes. Right now, I'm thinking – I just have to say it, because right now I am a little unsure about... because there are now so many decimals and... [indicates the final months with 2.21..., 1.99... and 1.79... nuclei] because one atom can't... you can't take 10 percent of one atom, like. So, this becomes sort of random whether, in a way... whether it splits or, like, if it loses one atom to radioactivity or not. So, I'm really not entirely happy with these numbers. But I can move on to the next one, I guess.*

We interpret this as Sophia revisiting her earlier statement: Can you have a fraction of a nucleus? This segment shows a lot of critique of her previous choice, which is indicative of sensemaking going on. Once again, we claim that Sophia engages in sensemaking as a response to the program output.

Sophia seems hesitant to exit the sensemaking process prematurely, and she may be experiencing some friction between the sensemaking and how she frames

## I. How Computation Can Facilitate Sensemaking About Physics: A Case Study

the interview situation. The initial "I just have to say it" at 20:18 seems to indicate that at that point she was about to engage in an activity she considered not wholly appropriate for the way she was framing the activity at the time (Russ et al., 2012).

We also note that in contrast to the previous sensemaking attempt, this one contains mainly physics ideas (atoms, radioactivity) with a nod to the mathematics embedded in them (percentages, probabilities).

### I.4.3 Sensemaking segment III

At this point the interviewer intervenes and invites Sophia to discuss a little more why she's not happy with the numbers, in effect sustaining the sensemaking frame. Initially this invitation is met with minor resistance. Sophia states that she doesn't want to spend so much time and energy thinking about an open-ended task that isn't clear about what it wants from her, so she's "choosing the easy way out". After being asked what she would do if she were a scientist and this was an important result to her, Sophia resumes the sensemaking process:

**Sophia [23:20]** *So, already after the third month here, then I would have taken, like, [indicates month 4 with 656.1 nuclei] here it reads point 1 – then I might have put in a for loop with choice? I think it is [random.choice()]<sup>1</sup> you use. Whether or not, like, that one... like, whether the decimal, whether that is a whole atom that goes away or not. So, in a way it becomes a sort of choice... thing. Such that when you run it as a model for the first time, then maybe... yes. Then maybe all... eh, the radioactive atoms are spent after, like, 56 months... and then the next time they are spent after 60 months. And the time after that maybe after 70 months. Eh, and then I would... yes, then I would have made a program or maybe a def-function and then run that many times and look at, percentage-wise, then, how probable is it that, eh, all the atoms... yeah, are gone after 50 months or after 70 months. So, I'd rather make that kind of model, because... eh, you kind of can't make this [indicates the output] completely accurate... But at the same time, when I think about it, it is... the probability of when that is going to happen is a little present in these numbers, too.*

At this point Sophia is using the language of *computing* as a tool for sensemaking, something that was absent in her earlier attempts. The mathematics and physics are still present in her argument. Sophia did mention randomness in segment II, but only here is she quantifying that randomness. She interprets these numbers (56, 60, 70 months) as probabilities of having 0 nuclei across an ensemble of *simulations*: "I would have made a program [...] and then run that many times and look at, percentage-wise, then, how probable is it that [...] all the atoms [...] are gone after 50 months or after 70 months."

<sup>1</sup><https://docs.python.org/3/library/random.html#random.choice>

The numbers that are printed by the program Sophia wrote are not probabilities of having 0 nuclei. But we note with interest that she states: *"when I think about it, it is... the probability of when that is going to happen is a little present in these numbers, too."* Sophia did not elaborate, but she is not wrong. The numbers printed by our simple program can be interpreted as the average number of nuclei left across an ensemble of simulations, and the chance that number is 0 in any given month will depend on that average<sup>2</sup>.

We claim that thinking in terms of writing code and the output that code produces is key for Sophia's bridging the gap in her understanding she has been wrestling with. As opposed to the simplified difference equation she was working with originally, the approach suggested here incorporates elements of randomness: two sets of 1000 nuclei would not necessarily decay in identical ways. This realisation does not mean she has a complete idea of how to implement it computationally, but sensemaking is about how you get there.

In summary, we have identified three sensemaking segments, where Sophia draws on knowledge from the following domains to explain the program output:

- Segment I: Mathematics
- Segment II: Physics (mathematics)
- Segment III: Computing (physics, mathematics)

These three segments together clearly demonstrate the sensemaking process: Sophia (a) realises that rounding the numbers hides information. It seems inaccurate that the number of nuclei appears unchanged for several time steps and then abruptly changes significantly more than 10%. But *not* rounding the numbers leads to working with fractions of a nucleus, which conflicts with her intuition about how the world works, as established prior to segment I. In each segment Sophia (b) iterates by proposing ideas and (c) critiquing these to make sure they are consistent in themselves and with other ideas.

The sensemaking process ends with the resolution of changing the interpretation of the numbers in the toy model. Instead of the actual number of nuclei in one experiment they represent probabilities across an ensemble of computational simulations. At this point Sophia has also attained a rough idea of how to implement the simulations in question.

## 1.5 Discussion and Conclusions

In this paper, we have shown that computing helped Sophia in two ways. First, she was able to modify her program back and forth between rounding and no rounding with relative ease. In the first two sensemaking segments, the inspecting and comparing the outputs of these approaches is her entry point into the sensemaking process: *"This looks a little strange..."*

---

<sup>2</sup>It is not calculable from that average *alone*, however. To calculate such a probability, one would need the probability distribution function, which in this case is the binomial distribution.

## I. How Computation Can Facilitate Sensemaking About Physics: A Case Study

Second, we argue that the key to Sophia's interpretation of the program output as representing probabilities is to think computationally about the problem, which is what happens in segment III. When discussing how to implement a more realistic model computationally, she realises that there is a connection between the code she is describing and her current output: "*The probability of when that is going to happen is a little present in these numbers, too.*"

We argue that this case study provides an existence proof that computing can provide fertile ground for students engaging in sensemaking. Specifically, working computationally allowed Sophia to (a) realise a gap in her understanding, (b) implement ideas and (c) test and critique the results for consistency. We observed that in this context, the idea that drew most heavily on computational knowledge (segment III) proved the most fruitful in the sensemaking process. Sophia uses computational language to frame her answer to the question of how she would approach the problem if thinking like a researcher. Framing the question in terms of the code she would write and the output it would produce allows her to make sense by connecting code to output, much as the computer would do.

This is last point is distinctly reminiscent of students linking problem types to counting outcomes in combinatorics, which students ordinarily have problems with. A study that expressed combinatorics problems in terms of conditional statements in Python found that connecting code to output allowed students to also connect problem types to problem outcomes. These results of that study suggests that thinking in terms of code and output may facilitate mathematical learning (Lockwood and De Chenne, 2020). The case of Sophia suggests the same in an interdisciplinary setting. Even though students may arrive at the correct answer without computing, their reasoning differs when students use computing.

Of course, one may question whether computing *causes* sensemaking in our example, and this is a fair question. Even though we have shown that Sophia initiates and sustains sensemaking in response to the program output, it is also possible that doing the calculations by hand could have produced the same results. We do believe, however, that Python's insistence that values are *typed*<sup>3</sup> influenced Sophia's thinking. When asked why she would round the numbers, she remarked that including the decimal point in *810.0* seemed unnecessary when the result was an integer. We think it unlikely that she would have seen similar output from a by-hand calculation (though of course her point that you cannot have half an atom still stands).

It is of course possible that Sophia's sensemaking was due to the task design or interviewer intervention. The former is not something we regard as a problem. If task design, rather than computing as such, should prove to be what caused Sophia to make sense of the activity, we have also demonstrated not only that it is possible, but also how. To a lesser extent that points also applies to interviewer

---

<sup>3</sup>The number *900.0* is a floating point number, while *900* is an integer, which Sophia learned in her programming class. Multiplying an integer such as *1000* with a floating point number such as *0.1* results in an implicit conversion of the former to a floating point number, which is what Sophia noticed when she printed these values.



intervention: asking Sophia to "think like a researcher" seemed key to have her frame the activity in a way that allowed sensemaking to continue. Requiring these kinds of interventions would be impractical for a large class of students, but it is possible that this framing could also be integrated into the task design to ease the demands on the teacher to intervene.

To determine under to what extent computing facilitates sensemaking, further research is needed. This would most likely require a large number of respondents and some research-based assessment task that produces evidence of whether students engage in sensemaking and whether they do so successfully. Such a task could, in this case, come in two versions, one with computing and one without, allowing for comparison of the groups of students.

In the other four interviews, we did note other examples of students beginning to engage in sensemaking in response to the output of their programs. What is special about Sophia's case was the way her computational resources helped her make sense of the apparent contradiction between the physics (realism) and mathematics (accuracy) in the model. It remains to investigate how this would play out in a classroom setting, where there is no interviewer to help sustain the sensemaking process like in Sophia's case.

Future studies could compare the thresholds for entering and successfully resolving a sensemaking process, respectively, using computing. This would have profound implications for how instructors integrate computing in science classes. If critical thinking is important to us, we should attempt to realise the full sensemaking potential in computational activities. It is then necessary to ensure that our students have sufficiently strong computational foundations to engage in these sensemaking tasks.

**Acknowledgements.** This study was funded by the Norwegian Agency for Quality Assurance in Education (NOKUT), which supports the Centre for Computing in Science Education.

## References

- Caballero, M. D., Kohlmyer, M. A., & Schatz, M. F. (2012). Implementing and assessing computational modeling in introductory mechanics. *Physical Review Special Topics - Physics Education Research*, vol. 8no. 2, 020106.
- Lockwood, E., & De Chenne, A. (2020). Enriching students' combinatorial reasoning through the use of loops and conditional statements in python. *International Journal of Research in Undergraduate Mathematics Education*, vol. 6no. 3, 303–346.
- Maloney, D. (2015). Teaching critical thinking: Sense-making, explanations, language, and habits. *The Physics Teacher*, vol. 53no. 7, 409–411.
- Odden, T. O. B., & Russ, R. S. (2018). Defining sensemaking: Bringing clarity to a fragmented theoretical construct. *Science Education*, vol. 0no. 0.

## I. How Computation Can Facilitate Sensemaking About Physics: A Case Study

- Russ, R. S., Lee, V. R., & Sherin, B. L. (2012). Framing in cognitive clinical interviews about intuitive science knowledge: Dynamic student understandings of the discourse interaction. *Science Education*, vol. 96no. 4, 573–599.
- van Someren, M. W., Barnard, Y., & Sandberg, J. (1994). *The think aloud method: A practical guide to modelling cognitive processes*. Academic Press, Inc.
- AAPT recommendations for computational physics in the undergraduate physics curriculum*. (2016). Retrieved July 6, 2018, from [https://aapt.org/Resources/upload/AAPT\\_UCTF\\_CompPhysReport\\_final\\_B.pdf](https://aapt.org/Resources/upload/AAPT_UCTF_CompPhysReport_final_B.pdf)

### **Authors' addresses**

**Odd Petter Sand** Centre for Computing in Science Education (CCSE), University of Oslo, Postboks 1048 Blindern, N-0316 Oslo, Norway, [oddsps@uio.no](mailto:oddsps@uio.no)

**Tor Ole B. Odden** Centre for Computing in Science Education (CCSE), University of Oslo, Postboks 1048 Blindern, N-0316 Oslo, Norway, [t.o.b.odden@fys.uio.no](mailto:t.o.b.odden@fys.uio.no)

**Christine Lindstrøm** UNSW Sydney, NSW 2052, Australia, [c.lindstrom@unsw.edu.au](mailto:c.lindstrom@unsw.edu.au)

**Marcos D. Caballero** Department of Physics and Astronomy, Department of Computational Mathematics, Science and Engineering, and CREATE for STEM Institute, Michigan State University, East Lansing, MI 48823, U.S.A. [caball14@msu.edu](mailto:caball14@msu.edu)





## Chapter 6

# The Path to Papers II and III

I now continue in describing my trajectory and the development of my research. This section describes how I went on to design the study that yielded papers II and III. Two standout features of the interview with Sophia influenced my subsequent choices of theories:

- Sophia connected knowledge across domains. Her computational reasoning first connects to the physical knowledge of decay (for each nucleus, decay is random), then to the mathematical model she had been working with (these numbers contain those probabilities). To better understand the finer elements of Sophia's sensemaking, I found that actor-oriented transfer (Section 3.2) would give us a language to describe how previous activity influence current activity.
- The task was designed to bring up an apparent contradiction: physically, we can only have an integer number of nuclei every month but rounding the numbers take away their explanatory power (no change for several steps, then suddenly much more than 10%)<sup>1</sup>. I attribute Sophia's sensemaking process at least in part to the task design that provided her with something to make sense of. The UbD framework (Section 3.1) might then allow us to describe the design features and principles involved, as well as the understandings Sophia came away with from the interview.

Both of these lessons were ultimately incorporated in the next task that I designed: a set of tutorials for the course *MAT-INF1100: Modelling and Computations*. This section serves as a bridge between my first paper and the final two. It describes the latter part of my work, from late 2018 until 2021, that resulted in these two papers.

As noted previously (in Section 2.1.2) MAT-INF1100 is a compulsory first-semester course for students in mathematics, physics, and electronics at the University of Oslo (Mørken, n.d.). This course integrates elements from both computing and mathematics and is taught alongside more traditional courses in calculus and programming. This triad of courses is intentionally coordinated.

While BIOS1100 (that I described in Section 5) made for an exciting context in which to study computing, mathematics and science complementing each other, I made the switch to designing learning activities for and interviewing students in MAT-INF1100 for the following reasons:

---

<sup>1</sup>If Sophia had insisted on using only integers in these calculations for the sake of physical realism, it would, interestingly enough, have led to a physically absurd result (we could never have gone below 4 nuclei). Insisting on mathematical accuracy, in contrast, does allow the number of nuclei to reach zero. That a realistic calculation (although invalid for small numbers of nuclei) leads to the *least* realistic answer is profoundly counterintuitive.

- BIOS1100 was a course in its infancy at the time and was changing rapidly from one fall semester to the next. This would have made an iterative design cycle spanning several years impractical in terms of comparing versions. In contrast MAT-INF1100 represented a more stable context for my research: the course itself changed very little, and changes to the tutorials would be able to explain more of the differences we might observe from one semester to the next.
- A course integrating knowledge from two domains (mathematics and computing) would be simpler to study than one integrating three (mathematics, computing and biology or physics), while still allowing for extension of my findings to other contexts later.
- Mathematics and computing are important for many fields in science education (Caballero et al., 2012; Malthe-Sørenssen et al., 2015; Reddy and Panacharoensawad, 2017; Weintrop et al., 2016). Thus, focusing on these two domains represents a higher potential for generalising my findings.
- I was invited to help teach BIOS1100 the following fall semester, and it would have been difficult, if not impossible, to combine this role with that of a researcher.

I designed three tutorials for MAT-INF1100. On the most fundamental level, these are centred on the following questions, that connect to the big questions from Section 1.4: How does one represent mathematical ideas in a computer program? And can one use math to represent computational ideas as well?

More specifically, a common thread running through all my teaching designs are ideas related to the representation of real numbers in the computer's memory. Few rational numbers can be represented exactly in the memory available to represent a number in the computer, let alone irrational numbers. *Rounding errors*<sup>2</sup> are important to be aware of for anyone using computers to model mathematics of science: even though they are often very small errors, they can accumulate and become very large if one is not careful<sup>3</sup> (Mørken, 2017).

Additionally, students may encounter *mathematical errors* that have little or nothing to do with how the computer represents numbers. One example of this is the error in approximations such as a Taylor polynomial, where the error can be quite large far from the point we expand around if we include too few terms. In numerical differentiation and to some extent integration, one will often run into both: with a too long step length, mathematical error will dominate, whereas one runs into accumulating rounding error if there are too many steps,

---

<sup>2</sup>Note that this represents a departure from the kind of rounding encountered in Sophia's task (Paper I). There, we were concerned with rounding to have our mathematical model accurately represent a physical phenomenon. The rounding errors we describe here, on the other hand, represent the fact that most numbers are impossible to represent exactly with a finite number of bits.

<sup>3</sup>I discuss this in greater detail in Section 2.2.

---

and these need to be balanced against each other. In both of these cases, it is important to be able to estimate and control the different types of error involved.

The overarching essential questions (see Section 3.1) that I ask the students across all my tutorials are:

- How do you know if an error is a rounding error or a math error?
- How do you balance the need for efficiency (speed) with the need for accuracy?

Table 6.1: Overview of the MAT-INF1100 tutorials and where they fit into the course schedule. The semester typically runs from late August to late November.

<b>Tutorial</b>	<b>Content</b>	<b>Position in course schedule</b>
1	Rounding errors Consequences for calculations	Week 3 (early September)
2	Taylor polynomials Optimal number of terms Logarithm functions	Week 8 (mid-late October)
3	Numerical integration Optimal number of terms	Week 12 (mid-late November)

The tutorials themselves, whose final versions can be found in Section 12.1, each focus on a particular aspect of computer representations and mathematical approximations of real numbers, shown in Table 6.1. Course material is introduced in lectures, and students work together in so-called *group sessions* under the supervision of a TA who answers questions that the students have. Typically, the material that appears in lectures one week is the topic of group sessions the week after that. Hence, "Week 3" in Table 6.1 refers to the third week of group sessions (the first lecture would be in week 0, which is perhaps appropriate for a course that incorporates this much computing). The midterm exam week, typically early October, is not counted as one of these weeks, as no teaching happens then.

The same set of interviews were used for the analysis of both Paper II and 3, as shown in Figure 6.1. Paper II uses data from all three tutorials to provide examples of the connections that students make between the domains of mathematics and computing. Paper III, in contrast, takes a more longitudinal view of one of the tutorials. I chose the Taylor expansion tutorial for this paper: initially, it was the tutorial that students struggled with, but after a thorough re-design it ended up providing the most interesting data in the second round of interviews.

It should also be mentioned that due to the COVID-19 pandemic, several interviews in the second phase had to be conducted digitally via the Zoom platform. For instance, if any of the students or the interviewer had any

## 6. The Path to Papers II and III

---

symptoms those persons could not be physically present. I also offered a digital interview if any of the participants felt uncomfortable with the risk of conducting the interview in person. As such, the interviews in the second phase (fall of 2020) were conducted in the following ways:

- A In-person interviews (3 of 7) on campus with video camera. Different students were designated to man the computer and whiteboard, respectively, so that at distance of at least 2 meters between participants was achieved throughout the interview.
- B Hybrid interviews (2 of 7) where the students were present on campus and the interviewer attended remotely via Zoom. This required an elaborate setup where one student shared the computer screen and the other filmed the whiteboard.
- C Digital interviews (2 of 7) where all the participants were in separate locations and shared their screens as necessary. In one interview I experimented with using a computer screen as a whiteboard, in the other one student had access to a physical whiteboard.

While these measures were successful in that we recorded no cases of COVID-19 connected to any of the interviews, only the data from category A were deemed usable. Category C interviews changed the way students worked in so many ways that any findings could not reliably be traced back to the tutorials themselves. Most notably, if the students used a keyboard to write math, it seemed to affect what they did in ways that presented difficulties for the analysis – the change of medium apparently presented a barrier for applying what they knew from traditional math, as they often interpreted these keyboard-written statements as computational instead<sup>4</sup>.

Nonetheless, even with access to a physical whiteboard, such as in the other category C interview and both category B interviews, the students appeared to be significantly less engaged in their work when the interviewer was not physically present. They spent less time on all parts of the task compared to the category A interviews (and those from the first phase) and seemed more concerned with completing the tutorial quickly. We speculate that the students may have been suffering from a form of "digital teaching fatigue" at this point, and comparisons between the interviews may indeed have been interesting, but as this is unlikely to be limited to mathematics and computing and at the same time quite likely to be studied at length by others in the education research community, we opted against making this phenomenon an object of analysis for the moment.

That said, this is the main reason I am only including one interview from the second round. On the other hand, this particular interview proved to be a goldmine of data for both papers. Supplemented with the breadth of data available from the first round, the overall impression is that we have enough

---

<sup>4</sup>This is interesting in and of itself, and even though we opted against focusing on this data due to its overall quality, we return to this phenomenon in Section 7.5.



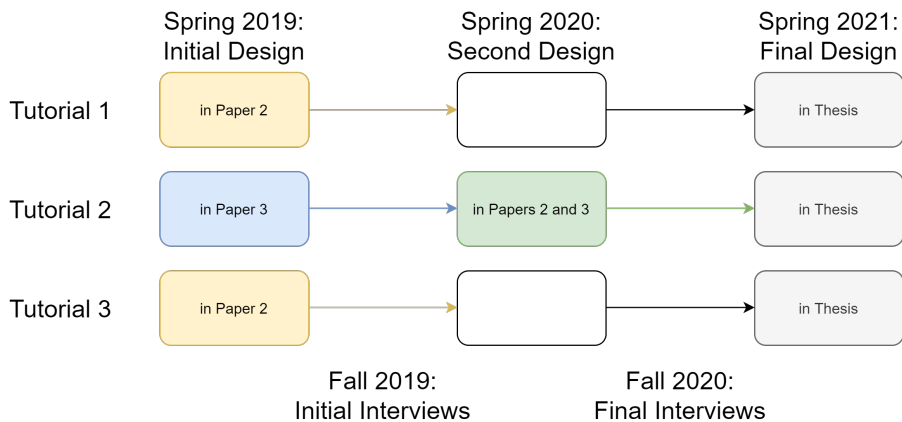


Figure 6.1: Distribution of tutorial versions across the final two papers.

data to investigate both transfer and tutorial design. Of course, it helps that we are not aiming to make broad, exhaustive claims of all possible ways students could work with mathematics and computing. Instead, both papers lay the groundwork for future studies by providing examples of what is possible.

A danger with such an approach in education research is that one tends to generalise based on extraordinary events. I hope to have avoided this by not making overly broad claims that my data cannot support. It may well be that some or even all of the students I interviewed are quite exceptional. Therefore, I do not claim that my findings demonstrate what is common; instead, I ask what is possible and what made it possible, with the implication that if we find these possibly extraordinary events desirable to recreate in the classroom, we have an idea how they came to occur. There might be barriers for most students to realise the potential I have outlined and identifying these would be a future research goal that complements my current work.

The most interesting episodes that I chose for the final analysis are summarised in Table 6.2. The final two papers are based on this data (see Paper II and III).

## 6. The Path to Papers II and III

---

Table 6.2: Students and interviews included in the final two papers. Gina, Benjamin, Martin, Lydia, and Roger were all part in more than one of the included interviews.

<b>Semester and tutorial</b>	<b>Students (pseudonyms)</b>	<b>Paper II</b>	<b>Paper III</b>
Fall 2019: Tutorial 1	Gina Benjamin	Yes	-
Fall 2019: Tutorial 2	Gina Benjamin Martin Ruth	-	Yes
Fall 2019: Tutorial 2	Lydia Roger Mathias	-	Yes
Fall 2019: Tutorial 3	Martin Lydia Roger	Yes	-
Fall 2020: Tutorial 2	Rita Lena	Yes	Yes

## Paper II

# Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

**Odd Petter Sand, Elise Lockwood, Marcos D. Caballero, Knut Mørken**

*Submitted for publication. Updated preprint available at <https://edarxiv.org/n8svc/>.*

### Abstract

This study uses actor-oriented transfer to investigate different ways in which students make connections across the domains of mathematics and computing. We interview first-year students at the University of Oslo as they work with a set of tutorials that we designed to integrate knowledge from both domains. The cases we present here demonstrate four different types of cross-domain connections: (a) mathematically reproducing the work of a computer program, (b) cyclically improving a program to produce better output, (c) coupling math to output to justify program improvements and (d) coupling math to code to justify program design. We provide rich examples of the ways in which students make these connections and discuss affordances for and barriers to mathematical learning in this context.

### Contents

II.1	Introduction . . . . .	63
II.2	Theoretical Perspective and Background Literature . . . . .	64
II.3	Methods . . . . .	68
II.4	Results . . . . .	76
II.5	Discussion and Conclusions . . . . .	104
	References . . . . .	110

### II.1 Introduction

The last several decades has seen computers take over more and more tasks that used to be the domain of the human mind alone. Already, there is a concern that

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

while mathematics is at the core of what computers can do, "the omnipresent mathematics is mainly hidden in all sorts of apparatus, which function as black boxes for its users" (Gravemeijer et al., 2017, p. 53). In mathematics classrooms, the worry is that students will become dependent on computational tools to do mathematics without understanding the underlying principles that allow these tools to do mathematics.

Nevertheless, computers and programming are becoming ever more important in the practice and teaching of mathematics (Broley et al., 2018; Passey, 2017). The reasons for this movement are manifold, and include allowing for the investigation of more and different topics, giving students more hands-on experience, and the envisioned potential "to have mathematics come to feel more natural, relevant, and less intimidating" (diSessa, 2018, p. 25). However, we as a field are just beginning to understand how the integration of computing into math affects student thinking and learning. How does one domain connect to the other from the students' point of view, and what do these connections afford?

Such connections have long been the territory of the study of transfer, which is seen as a fundamental goal of the existence of educational institutions (Billett, 2013). However, traditional transfer studies' focus on *correctness* means they tend to miss many of the generalisations that students make (see Section II.2.1). With the *actor-oriented* transfer perspective, the students' point of view is put front and centre, allowing for a more detailed account of students' generalisations (Lobato, 2008).

Our focus in this paper is to study the ways in which undergraduate students connect mathematics and computing. While studies of undergraduate students' mathematical work using computing do exist (Section II.2.2), there have been calls for more attention to be paid to computing in math education research (e.g., (Lockwood and Mørken, 2021)). We therefore aim to provide several rich examples that demonstrate how working in a context where mathematics and computing are integrated can be beneficial for the students' thinking and learning about mathematics.

To that end, we first present our theoretical perspective and position our study within the literature, before formulating our research questions (Section II.2.3). In Section II.3, we describe the study design, the process of analysis and the analytical framework we employed. Finally, we present our cases (including brief descriptions of the tutorials we designed) and our analysis of each in Section II.4, and discuss these results and their implications for teaching and further research in Section II.5.

## II.2 Theoretical Perspective and Background Literature

### II.2.1 Theoretical Perspective

*Connections* and their importance in the context of learning was highlighted as early as the 19<sup>th</sup> century. Høffding, 1892 argued that "what matters is how the new situation is connected with the thinker's trace of a previous situation, which may be quite idiosyncratic" (cited in (Lobato and Siebert, 2002)). While

there are many different types of situations students may encounter as they work with computers in mathematics and science (see for instance Weintrop et al., 2016), the ways in which students connect their work on the computer with mathematical concepts is largely unexplored territory (the examples of previous work we found are cited in Section II.2.2).

While this paper is not focused on transfer of learning per se, the *actor-oriented transfer* (AOT) perspective (Lobato, 2012; Lobato and Siebert, 2002) provides us with a useful language to describe these connections. An important facet of the AOT framework is its focus on *similarities* as seen from the student's point of view. These similarities allow actors to make connections between the activity they find themselves in and some previous activity that they regard as similar and relevant. In (Lobato and Siebert, 2002), "[AOT] is defined as the personal construction of relations of similarity between activities, or how 'actors'<sup>1</sup> see situations as similar." This applies even when students make unexpected connections that may or may not result in incorrect performance, and stands in contrast with traditional views of transfer (Lobato, 2008).

An example of a study grounded in AOT that explores connections in a mathematical setting is that of (Karakok, 2019), who looked at physics students' connections among representations of eigenvectors. In this context, Karakok reports on "the necessity of developing flexible shifts between different modes of thinking" and links this to the development of "instructional materials and interventions that emphasize opportunities for students to inquire and *connect multiple modes of thinking*" [emphasis added]. These are the kinds of connections we are looking for between computing and mathematics.

The modes of thinking Karakok describe correspond to different *representations* of eigenvectors that correspond to points of view which each has different affordances for the students. The term *affordances* has often been used in the mathematics education research literature, most notably in the context of technology. The term describes a relationship between an actor and an object that is expressed by some activity: an illustrative example by (Gibson, 1979) is that water affords drinking and drowning to human beings, while it affords breathing to fish (Brown et al., 2004). In our context, when we discuss affordances and limitations, we are concerned with the activities that are helped or hindered, respectively, as a result of interactions between humans and computers.

(Greeno et al., 1993), one of the inspirations for the AOT perspective, discussed affordances in the context of *reasoning* with representations. This use of the term is notably different from discussing the affordances of computational tools in themselves:

Representations include symbolic expressions that represent actual or potential states of affairs. Representations also include physical constructions such as diagrams, graphs, pictures, and models with properties that are interpreted as corresponding to properties of situations. [...] Affordances for reasoning, on this account, are

---

<sup>1</sup>In our case, the actors are students.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

properties of representations in relation to a person's or group's abilities to use the representations to make inferences. Reasoning is an activity that transforms a representation, and the representation afford that transformational activity. Abilities for reasoning activities include knowing the operations to perform on the notational objects in the representation and understanding the semantic significance of the objects and operations. *Conceptual reasoning* occurs when representations of concepts are included in the representations that are used in reasoning. [...] Generally, concepts correspond to properties or relations in a domain (pp. 108-109).

Greeno goes on to present algebraic or arithmetic representations as examples that present affordances for reasoning within the domain of mathematics. This paper aims to investigate the affordances of *cross-domain* reasoning, across the domains of mathematics and computing. To that end, describing the connections (or, in Greeno's terms, the reasonings) that translate representations from one domain to the other is of interest.

It should be mentioned that the concept of *domains* is somewhat contentious<sup>2</sup> and points to bigger questions than we have room for in this paper. While it is certainly worthwhile to investigate and discuss which knowledge belongs to a particular domain, our research questions (see Section II.2.3) implicitly assume that the domains of mathematics and computing exist, if only as arbitrary organising principles. We do not claim that any particular piece of knowledge is *inherently* mathematical/computational, nor do we suggest that there is no overlap between domains. In fact, if there were no overlap at all, we expect that any connections we identified would be both rare and rather contrived. Our data, as we shall see, suggests otherwise.

In this paper, we will analyse how university students make connections and interpret affordances when working with computer programming to solve problems that integrate elements of mathematics. Henceforth, when we use the term "computing", we will follow (Lockwood and Mørken, 2021), who consider *machine-based computing*, i.e. "the practice of developing and precisely articulating algorithms that may be run on a machine". Applied to our context, this means that the students use Python to solve problems that are both mathematical and computational.

The computational representations that our students encountered were mainly of two types: the source code they were working with and the various outputs that code produced when the students ran their programs. We do not claim that these are the *only* possible representations or activities within machine-based computing (and certainly not within computing in general), but our data and interest in classifying connections suggest that this distinction is useful for our purposes. It is entirely possible that different, perhaps finer, distinctions are useful in other contexts.

---

<sup>2</sup>For instance, Inagaki and Hatano, 2002 spend some time questioning the conventionally assumed nature of domains in their work on the development of biological knowledge, while (diSessa, 2017) aims to show that at a fine grain size, domain boundaries appear to be false.

## II.2.2 Relevant Literature

Having established our theoretical perspective, we will proceed to frame this study within existing literature in mathematics education research. More precisely, our study features among those that explore connections between mathematics and computing at the undergraduate level. Such studies most often explore a particular subdomain of mathematics, for example combinatorics (Lockwood and De Chenne, 2020) or statistics (Ramler and Chapman, 2011). Our examples in this paper span several such domains, such as Taylor series, logarithms, and integration. There is a common theme, however: the computational representation of real-valued variables and functions.

The perceived value of computing in mathematics dates back to the work of Papert, who proposed that the computer offered unique affordances for students learning mathematics. Using the *Logo* programming language designed for this purpose, Papert studied children's discovery of algorithmic ways of thinking about geometry through use of the computer. The classic example of this is children writing programs that move a Turtle across the screen, tracing out geometrical shapes in the process (Papert, 1993).

A more recent example that uses the AOT framework in the context of computing is (Lockwood and De Chenne, 2020), which explores how students use Python to list and count the outcomes of combinatorial counting problems. One way to achieve this is using conditional statements to eliminate outcomes that do not respect restrictions on ordering, repetition, or both. Lockwood and De Chenne found that this approach focused students' attention on the outcomes they were counting, which again reinforced the conceptual differences between different types of counting problems.

There are also examples of studies that use other frameworks or that cover several mathematical subdomains; (Buteau et al., 2016) is an example of both. In this paper, the authors present a case study of a single student spanning three semesters and the student's work on 14 assignments over these semesters. These assignments are connected to many different mathematical topics, and the authors focus on her learning experience across these semesters. This study differs from ours in that they investigate courses designed from the ground up to provide these experiences - our tutorials, on the other hand, are designed to complement a pre-existing mathematics course (see Section II.3.1 for details).

Going beyond the undergraduate context, there are numerous examples of studies exploring the interplay between mathematics and computing for younger students. Ilana Lavy used Logo to investigate the different types of mathematical arguments such students construct when working in a computerised environment (Lavy, 2006). More recently, Benton and colleagues (Benton et al., 2017; Benton et al., 2018) designed interventions using Scratch to have students reason with geometrical concepts and place value. (DeJarnette, 2019) also used Scratch to investigate the difficulties that schoolchildren face when trying to understand the meaning of symbols and how they fit together. (Kaufmann and Stenseth, 2020) had young students program in Processing to solve mathematical problems, and found that their mathematical arguments became more elaborate over time, but

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

that the trial-and-error method the students often used might counteract this effect to some extent.

The context we are examining (Section II.3.1) is one in which mathematics and computing are taught in an integrated fashion. We investigate students working with knowledge from these two domains in the same setting it has been taught, which does not require proof of transfer to a *different* context. The actor-oriented transfer that we investigate occurs between domains, but the context of our tutorials is not fundamentally different from the one that our students are familiar with from their classes.

This paper aims to provide several examples of how students use computing and mathematics together. These examples are mainly related to the representation of real numbers in computers, and we view them through the lens of student connections and generalising activity that we presented in Section II.2.1. To this end, we will present a framework in Section II.3.2 that allows us to describe the different ways in which students make connections across the domains of mathematics and computing.

### II.2.3 Research Questions

The research questions of this paper are as follows:

1. What types of cross-domain connections do students form between mathematics and code/output?
2. What do each of these connections afford the students?
3. Which patterns of connections emerge in the students' work?

Research Question 1 will be the focus of the theoretical framework in Section II.3.2, which was constructed based on the analysis of the data. The cross-domain connections also emerge and are exemplified in Section II.4, where we apply the framework to examples from the interview data. Research Question 2 will also be treated in Section II.4, while Research Question 3 is addressed in Section II.5, where we summarise the interviews based on the answers of the first two questions.

## II.3 Methods

### II.3.1 Data Collection

We collected data in two phases: a first phase in the fall of 2019 and a second phase in the fall of 2020. In both semesters, we recruited students from the course *MAT-INF1100: Modelling and Computations*, a compulsory first-semester course for students in mathematics, physics, and electronics at the University of Oslo, Norway. This course integrates elements from both computing and mathematics and is taught alongside courses in calculus and programming (Mørken, n.d.). As this course appears at the beginning of the students' undergraduate program, it



is taught in Norwegian. We conducted the interviews in Norwegian and, later, translated relevant parts to English for joint analysis and publication.

We designed three tutorials that were used both for the interviews and regular group sessions in the course. Our use of the term "tutorials" is in the sense that students mostly work independently, but have access to the assistance of teaching assistants (TAs) in these group sessions at need <sup>3</sup>. In the interview sessions the interviewer performed this role when the students got stuck or requested it. We summarise each of these tutorials in the introduction to the corresponding case in Section II.4.

### II.3.1.1 First Phase

In the first phase, we initially recruited volunteers using an online form in a lecture and grouped their answers according to time availability, gender, previous programming experience in general and in Python specifically. Based on the students' responses, we assembled 3 pairs with different gender configurations and levels of experience. Unfortunately, only one student from each pair showed up to each interview, which necessitated using a think-aloud protocol (van Someren et al., 1994) to ensure that we got somewhat useful data from these individual interviews. In the end, we did not use this data for the analysis, however, as the group data proved richer and closer to the classroom setting.

We assembled a fourth group (Case A in this study) to get at least one group interview for Tutorial 1 and started inviting 3-4 students to each interview to account for no-shows. During classroom observations, we became aware of some groups of students working together in ways we thought would provide interesting interview data. We therefore recruited additional groups of students in this way. This second set of students had the advantage of having worked together previously, and insofar as they were all available at the same time, we tried to interview them together for the last two tutorials.

All in all, in the first phase we interviewed 13 students in 8 interviews: four interviews with Tutorial 1 and two each with the last two. Five of the students participated in two interviews (the three students in Case C were interviewed with Tutorial 2 and Tutorial 3). In the cases we selected for inclusion in this paper (Section II.4), we will describe the participating students in the beginning of each case.

The first author conducted all the interviews in person using both video and screen recording on a computer where a familiar Python programming environment had been set up for the students to use. They also had access to a whiteboard, which the video camera captured. The students were told to work together as they would in a normal group session in the course, using the interviewer as a TA where necessary. The authors collectively decided on follow-up questions to gather feedback for the next design cycle and record their previous experience with programming.

---

<sup>3</sup>It also evokes the format that inspired the early versions: the Maryland Tutorials (Steinberg et al., 1997)

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

The interview protocol was set up ahead of time, inspired by (van Someren et al., 1994) even for the group interviews. It reminded the interviewer to intervene only when students were stuck for an extended period of time or if they explicitly asked for it (the students were also informed about this at the beginning of each interview). This was done to ensure that the students' own ideas were not *needlessly* affected by interviewer interventions, even though such interventions proved difficult to avoid entirely.

The sole exception to this instruction was that when students did something interesting, the interviewer could ask them to elaborate on their thinking. While these elaborations might well affect the students' thought process, the interviewer would not ask specific questions that could reveal the interviewer's thinking, as could be the case when the students asked for help or got stuck. Follow-up questions were devised ahead of time, though interesting event during the interview could also be the topic of such questions. In these cases the interviewer would write down the question and return to it once the students' work with the tutorial had finished. This reflects the trade-off between not wanting to influence the students' work versus the possibility of inaccurate answers to questions posed so long after the fact.

### II.3.1.2 Second Phase

In the second phase (fall of 2020), we once again recruited students using an online form, this time asking them to specify when and where they were scheduled to have group sessions. This was done for three reasons, which were (a) to interview students that were used to working together, for the sake of authenticity, (b) to comply with COVID-19 protocols and ensure minimal mixing of cohorts, and (c) for ease of scheduling. The interviews were conducted at the same time as the ordinary groups worked on the same tutorial; hence, the students were likely to be available for interviews at that time.

In the registration form, we asked the students to volunteer names of other students they preferred to be interviewed with, and we complied with their preferences where possible. Observations from the first round of interviews suggested that students who knew each other from class had an easier time participating actively and verbalising their understandings and non-understandings.

In this phase we interviewed a total of 7 students in 7 interviews: two pairs of students were interviewed with all three tutorials (Case B represents one such pair), whereas the remaining three students declined further interviews after Tutorial 1.

The interview protocol remained largely unchanged in the second phase, except that the follow-up questions in the second semester targeted how students perceived using mathematics and computing in an integrated way, as well as mapping their programming experience as before. An important exception is the interviewer giving two students an extra challenge not originally part of the tutorial. This on-the-fly change was not planned ahead of time, but ended up producing very interesting results (see Section II.4.2).

## II.3.2 Analysis

In this subsection, we first present the analytical framework that we developed to characterise the students' connections. Then, we describe the process of developing and applying this framework to our analysis of the interview data.

### II.3.2.1 Analytical Framework

In this project, our focus is on student connections that straddle the domain boundary between computing and mathematics. We would like to note that it is not completely trivial to identify what is a mathematical activity, what is a computational one, and what is both. As a starting point, we assigned confidence levels to these labels, and the first author then grouped the high-confidence segments together to see what they had in common. This work became the foundation for our analytical framework for labelling connection types.

We distinguish between two aspects of the domain of computing: Code (programs the students write in the code editor) and Output (terminal window printouts and plots produced by these programs), as we observed that students tend to interact differently with each of these aspects. We take the Code as a mutable set of instructions for the computer to interpret literally, whereas the Output is a result of the computer's activity. The students can only affect it indirectly, by altering the Code or other input data.

After categorising relevant segments in our data, we developed a classification scheme to label these connections. These labels became a useful lens for us to view the data through in the process of analysing them.

To assign a label to a connection, we focus on the four properties outlined in Table II.1. Some of these properties mirror the discussion of symbolic notation found in (DeJarnette, 2019). In particular, Target Syntax and Target Structure are reminiscent of "vocabulary" and "grammar", respectively: the former describes the meaning of specific symbols, whereas the latter refers to the way the symbols are connected to communicate meaning.

For a connection, each of these four properties can be valued Code, Output, or Math. In both Paper I and Lockwood and De Chenne, 2020, there are signs that students interact with Code and Output differently. The former can be manipulated directly, and the latter only indirectly, through making changes in the former. The Output can also visualise a great number of data points produced by the Code, and reveal patterns that are not evident when writing the program. The Code, on the other hand, can make visible the algorithms or the logic behind the program in a way that the output often obscures. We therefore find it meaningful to distinguish between these two subdomains of Computing.

We will exemplify these ideas in the results, but present here a contrived but illustrative hypothetical example: imagine a group of students making a connection in speech to a program that they remember making (Source Domain: Code) as they are writing "cat = 42" on a whiteboard (Target Medium: Math) and describing this as an equation (Target Structure: Math). From the students'

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

Table II.1: Properties used for labelling connections.

<b>Property</b>	<b>Meaning</b>	<b>Evidence</b>
Source Domain	The previous activity students reference as they make the connection (from where is something being transferred?)	Verbal utterances Observing written work
Target Medium	The physical/visual medium students work with as they make the connection	Observing actions Verbal utterances
Target Structure	The ways in which students structure or organize pieces of information in the Target Medium	Verbal utterances Observing written work
Target Syntax	The ways in which students represent and interpret the pieces of information themselves	Verbal utterances Observing written work

conversation it is clear that they interpret the piece of information "cat" as the product of three variables, c, a, and t (Target Syntax: Math).

Other possible interpretations that do not apply in this example could have been interpreting "cat = 42" as a variable assignment (Target Structure: Code), or as something that could be printed to the terminal window by the program (Target Structure: Output).

A cross-domain connection such as the one in the example above is recognised by the Source Domain and the three Target properties belonging to different domains. We consider Code and Output to both belong to the domain of computing, while Math is in a domain of its own. There are four types or labels that we have assigned to the connections we found in our data, see Table II.2. We acknowledge that this is not an exhaustive list, but an account of the kinds of cross-domain connections that we observed in our interview data.

We label a connection (1) Math Implementation when the Source Domain is Math, and the Target properties are all Code. An example of Math Implementation is when the students write a loop in code to calculate the sum of a series of numbers that follows a particular pattern.

We label a connection (2) Code Modelling when the Source Domain is Code, and the Target properties are all Math. An example of Code Modelling is when students produce a mathematical proof of concept for a piece of a program.

Table II.2: Connection labels.

Label	Source Domain	Target Medium	Target Structure	Target Syntax
Math Implementation	Math	Code	Code	Code
Code Modelling	Code	Math	Math	Math
Output Modelling	Output	Math	Math	Math
Mathematical Interpretation	Math	Output	Output	Output

We use the label (3) Output Modelling when the Source Domain is Output, and the Target properties are all Math. An example of Output Modelling is when students make use of mathematics to determine the required changes to a program that makes it produce some desired output (instead of resorting to trial and error). Where Code Modelling is mainly associated with designing or attempting to understand programs, Output Modelling is instead associated with debugging or refining a program that is already implemented and reasonably well understood.

Finally, we have (4) Mathematical Interpretation, meaning that the Source Domain is Math, and the Target properties are all Output. An example of Mathematical Interpretation is when students call on mathematical knowledge as they are engaging with the output of a program. For instance, they could be verifying that the Output makes sense mathematically or making informed decisions on how to improve the program going forward. Unlike Output Modelling, they are attending to plots, tables or files produced by the program instead of the Math itself.

Note that while Table II.2 might suggest an insistence that all the Target properties must be identical, we do allow for exceptions when there is supporting evidence that allow us to assign a label with confidence. For the data presented in this paper, that was rarely the case. In some cases, the Target properties instead painted a hybrid picture. On two occasions, we came to interpret these hybrids as half-formed connections that the students returned to and connected fully later on<sup>4</sup>.

In Figure II.1, we illustrate the labels from Table II.2 as connections originating from the Source Domain to the Target activity.

---

<sup>4</sup>While these "half-connections" are not our primary focus in this paper, they are still relevant, as a pair of them can be interpreted as a full connection happening in two distinct stages. As an example, consider the stages of a connection that we will examine more closely shortly, in Case A:

Source Domain: Code → Target properties: Math/Code hybrid  
Source Domain: Math/Code hybrid → Target properties: Math

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

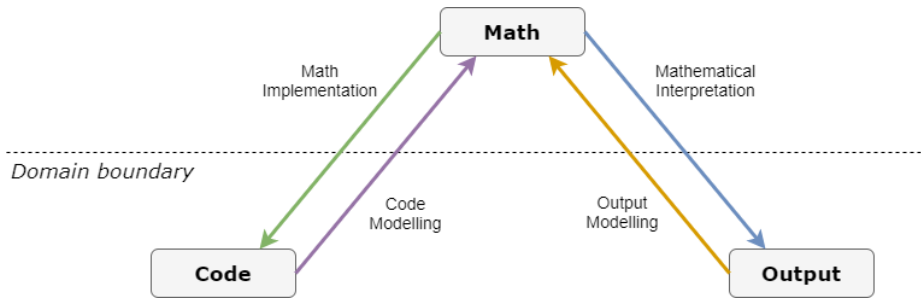


Figure II.1: Cross-domain connections. Note that the name of the label corresponds to the starting point of the arrow, which represents the Source Domain.

### II.3.2.2 Data Analysis

This paper constitutes an exploratory case study, and no pre-existing code book was available for use in the analysis. We therefore performed a *thematic analysis* of the data, following roughly the six-phase process outlined in (Nowell et al., 2017):

1. Familiarising yourself with your data
2. Generating initial codes
3. Searching for themes
4. Reviewin themes
5. Defining and naming themes
6. Writing up the report

In phase 1, the interview audio was transcribed at the end of each semester. I reviewed the transcripts to divide them into short segments with brief descriptions of each. Segments were flagged for further review if they involved both mathematics and computing, and the students' work was focused on understanding something, as opposed to performing a skill or recalling simple facts.

This selection process produced several episodes (usually consisting of several consecutive segments) where students either (a) were engaged in some activity that contained elements of both computing and mathematics, (b) made a transition between computing and mathematics, or (c), both. The first author then translated the transcripts from Norwegian into English and enhanced them with evidence from other sources, such as screen and video captures, photos of the whiteboards and collected worksheets. These enhanced transcripts were shared with the rest of the research group (the co-authors) and formed the basis of the analysis from that point onward.

In phase 2, we divided each episode into short segments and coded each segment. We then looked at the bigger picture, focusing on the patterns that were apparent from the students' point of view. Based on this, we formulated claims supported by evidence from the transcripts, in analytical memos. The co-authors reviewed the claims and their justifications for the sake of validation and helped refine the claims in several steps. Examples of these early codes showing our interest in the integration between mathematics and computing were: pure coding, pure math, integrated. We also attempted to describe different subtypes of integration based on what was in our data, arriving at early versions of the figures in Section II.5.1.

For phase 3, we assigned labels to the episodes, identifying and highlighting key actions that we took as justifications for this coding. Examples of these include verbal utterances from the students, things they wrote in the code editor or the whiteboard, and other actions. These key actions are listed in the "Evidence" column of Table II.1.

Initially, we made a broad pass and subsequently selected the most promising episodes, focusing on justifying why these episodes should be a priority. Cases A and C in Section II.4 came out of this pruning process. In the process we kept detailed notes of each version of these descriptive labels and triangulated the labels with the co-authors. Examples of early themes or labels were: "mapping code to mathematics", "mapping mathematics to code" and "explicitly telling the code what to do".

Phase 4 involved creating new enhanced transcripts with a separate column for coding and key actions to get a clearer picture of the codes as applied to raw data, including data we had not previously analysed in detail. In the process, we also linked the key actions to theory. For example, we noticed that a lot of our key actions were students making connections or noticing similarities.

After several iterations, we decided to focus on cross-domain connections alone, ignoring connections that belonged exclusively to one of the domains (all mathematics or all computing). In the AOT literature we found nothing about connections between domains in general, even though (Lockwood and De Chenne, 2020) is another example of the same.

In phase 5 we summarised what each of the selected episodes demonstrated and refined our coding of these based on this, before going back into the data and re-coded them using the updated labels. In the process, we collectively decided on names for the labels in several steps. The result of this process was an early version of the diagram in Figure II.1.

Realising that we had to define these arrows/labels more clearly, the first author suggested names and descriptions for each, and the other authors contributed to the discussions about how to define these. During these iterative discussions, the four properties Source Domain, Target Medium, Target Structure, and Target Syntax emerged, as shown in Table II.1 The pattern described in Figure II.20 was also identified at this time, as well as the half-connections described in the previous subsection (see footnote II.3.2.1, plus Case A and B in Section II.4).

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

For the final phase, we started writing the papers, initially focusing on describing the themes and the process leading to their development. We referred back to the theoretical framework to justify our choice of themes, and attempted to articulate what each theme meant and revealed about the topic. This appeared in the form of patterns of connections and what each pattern afforded the students (a step up in grain size).

At this point, we also had available transcripts from the second phase and highlighted episodes from this second round of data, where we recognised that the emerging framework could be used for analysis. Case B was identified at this stage, as well as a new label (Output Modelling) that had not emerged in the first phase data. We finally sent finished drafts of each paper to all respondents to establish the fit between their views and our representation of them.

### II.4 Results

This section contains our three cases, one from each tutorial. We will describe the students at the beginning of each case, after we provide a summary of each tutorial as necessary background. Cases A and C are from the first phase (fall 2019), while Case B is from the second (fall 2020). All three tutorials went through design changes between the two phases, but only Tutorial 2 underwent a complete redesign and change of focus as a result of the interviews and feedback from the students. The other tutorials were refined for clarity and to make better use of the students' time. We expanded the tasks that we identified as central to the tutorials' learning goals and removed several tasks that were time-consuming but involved little conceptual understanding.

#### II.4.1 Case A: Gina and Benjamin (Tutorial 1, 2019)

In our first case, we focus on two students, Gina and Benjamin. This case highlights the ways in which these students used mathematics on a whiteboard to model a piece of code and in the process, they constructed a proof of its correctness. At that point, the students' focus shifted to the output produced by the program, where they sought to explain it based on their mathematical considerations. This, using the framework of Section II.3.2, constitutes an example of Code Modelling followed by Mathematical Interpretation.

##### II.4.1.1 Tutorial 1: Rounding Errors

Before we present excerpts of the interview followed by our analysis, we provide a summary of the tutorial as necessary background, as we will do for all the cases under consideration. In the first tutorial, the students were presented with a short program designed to take a randomly generated number between 0 and 1, and then calculate what number (the step size) must be added to the original random number 9 times to get exactly 1.0 as the result (see Code Sample II.1):



```
from random import random
x0 = random()
sum = x0
diff = (1 - x0)/9
while sum < 1.0:
    print(sum)
    sum = sum + diff
print(sum)
```

Code Sample II.1: The program from Tutorial 1, written by the first author.

In practice, however, the program did not produce 1.0 every time, even though the code contained a *while* loop that kept running as long as the sum up to that point was less than one. The point of this task was to raise the issue of rounding errors, as a rounding error in the sum would frequently cause the loop to continue for an additional iteration beyond what was intended, leading to a large error in the final answer.

We tasked the students with testing the code and finding out what, if anything, was wrong with it. At the end of the tutorial, they were given a set of exercises to help them understand how the computer's binary representation of real numbers determine where rounding errors will appear and where they will not.

#### II.4.1.2 Case A: Interview and Analysis

Gina and Benjamin were first-year students, neither of which had any previous experience with programming prior to the semester. By the time of the first interview, they had two and a half weeks of Python experience. Gina self-identified as a "problem solver" and tended to verbalise her thinking, including working theories and understandings. Benjamin tended to speak somewhat less than Gina and would typically comment on the work Gina did on the computer or whiteboard. At times, Benjamin would take over the computer or whiteboard and direct the conversation.

Gina and Benjamin had spent some time discussing what the code did, and they ran the program a few times. Subsequently, they used the whiteboard to mathematically describe an imagined case where the random number was 0.55, and the resulting step size 0.05. They predicted what the program would print in such a case, and Gina noted explicitly that she recognised this as what she referred to as uniform spacing.

The students then focused on the code for a while, trying to debug it by looking for logical errors<sup>5</sup>, such as using the wrong variable at the start of the calculation. There was no evidence of mathematical reasoning in this process. Unable to find any such errors, the started discussing what the piece of code was supposed to accomplish, but soon began to express some hesitation and doubt.

At this point, Gina turned to the whiteboard and wrote the expression in Figure II.2, presumably to help move the process along:

---

<sup>5</sup>An error that does not produce an error message. In the presence of such an error, the program can run, but it does not produce the intended result due to faulty logic.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---



The image shows a handwritten mathematical expression on a whiteboard. The expression is  $x0 + \text{diff}_1 + \dots + \text{diff}_9 = 1.0$ . The variables  $x0$  and  $\text{diff}_i$  are written in a cursive, handwritten style. The equals sign and the constant  $1.0$  are also handwritten.

Figure II.2: Gina's first expression.<sup>6</sup>

**Gina** [while writing] *So, we've got x, eh, plus, no...* [looks over at worksheet] *was it plus or was it times? We were supposed to add another number* [resumes writing] *plus diff, eh...*

**Benjamin** *times 9?*

**Gina** [keeps writing] *9 times, we were sort of supposed to... 1 plus blah blah blah, and up to diff<sub>9</sub>, eh...*

**Benjamin** *Yes, then all of them are equal to [inaudible].*

**Gina** *And all of this together should become 1.*

We suggest that this change of activity represents the beginning of a cross-domain connection from Gina and Benjamin's point of view. We have labelled this connection as an instance of Code Modelling (see Table II.2). Interestingly, this connection happens in two stages, where the exchange above marks the starting point. A little later in the interview, we will see the students make a further connection from this work to something they regard as fully mathematical, completing the Code Modelling. But first, we present our justifications for the labelling applied here.

We identify the Target Medium as Math from Gina's writing on the whiteboard in Figure II.2. The Source Domain (the idea being represented) is Code because her expression models the mathematical operation that the code is performing on the left-hand side, and the desired outcome on the right, pointing back to their engagement with the code just moments prior. While the Target Structure in Figure II.2 certainly looks like Math to an observer, we shall see that both Gina and Benjamin displayed excitement later, when they realised that this statement could in fact be interpreted as an equation. It is therefore problematic to claim that the Target Structure was Math from the students' point of view *before* they made that realisation.

The Target Syntax is an interesting hybrid between Code and Math, containing some elements of each. We recognise Code syntax in the variable names, as  $x0$  and  $\text{diff}_1$  are identical to the variable names in the code and are unconventional ones to use in mathematics. We would consider  $x$  and  $d_1$  to be more traditional choices.

Furthermore, in mathematics it would be considered unnecessary to write the constant 1 as 1.0, whereas in computing, 1 and 1.0 are values with different data types (integer and floating-point number, respectively). Gina said 1 aloud but

wrote 1.0, and we infer that meant this as a computationally typed representation (1.0) of a mathematical value (1). We do note, however, that their decisions to write the numbers in this way may have been influenced by their appearance in Code Sample II.1, where they do indeed have different data types.

Math syntax was also present: Gina wrote the while loop as a sum (because the loop adds one term each iteration), using subscripts 1 and 9 to label the different iterations. The use of "... " on the whiteboard to mark repetition is a conventional mathematical way of describing the sort of repetition the loop represents, and her saying "blah blah blah" as she wrote it implies that she meant it in that way.

We interpret the equality to stand for mathematical equivalence and not computational variable assignment from the students' point of view, because throughout the interview the students demonstrated that they knew that the latter requires a single variable on the left-hand side to store the value on the right. Nonetheless, we do not claim that the students thought of the Target Syntax as Math at this point - even though this hybrid syntax laid the groundwork for their subsequent interpretation of the expression in Figure II.2 as something mathematical.

In sum, we interpret this event as the students writing a pseudo-mathematical expression on the whiteboard that they regarded as similar to the code. We find it interesting that they did so on their own accord, without prompting.

Gina went on to say:

**Gina** *Okay, you can surely write a, that is, by hand we can surely write something sum of [inaudible].*

**Benjamin** *Yes, that is, because all those diffs are equal then.*

**Gina** *Those diffs are equal.*

**Benjamin** *So it is really...*

**Gina** *So it is, but the diffs are in a way dependent [connects  $x$  and  $diff_1$  with a curved line as seen in Figure II.3], that is, these are connected because the expression for  $diff$  is 1 minus  $x$  [writes the statement in Figure II.4].*

**Benjamin** *Yes, but that happens outside the loop.*

**Gina** *Mm-hmm, that happens outside the loop, yes.*

**Benjamin** *So it is just calculated once.*

Gina's choice of words "by hand" emphasises that she sees the Target Medium as Math-related. Her mention of writing something as a "sum" could indicate that she does not regard the Target Structure as being Code at this point. The Source Domain could still be Code, however, if she was thinking of the loop being *represented* by a sum sign, which is something the students were likely

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---



Figure II.3: Line connecting  $x$  and  $diff_1$ .

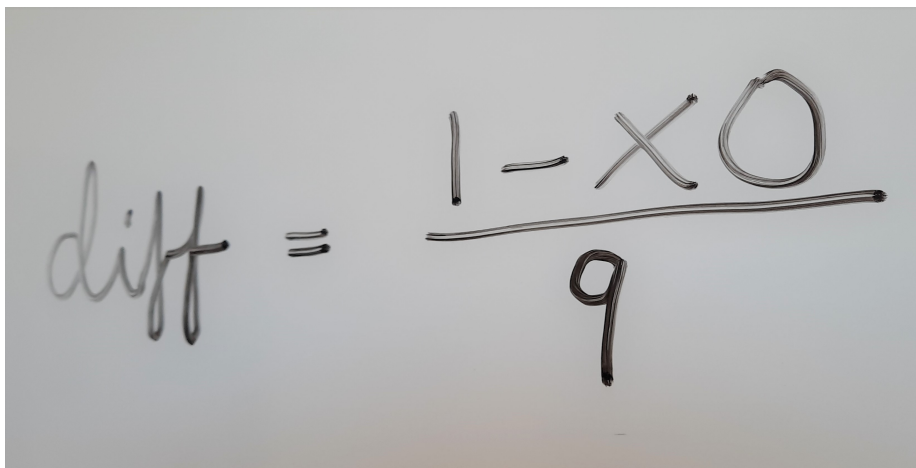
A photograph of a whiteboard with handwritten text. The text reads "diff = (1 - x0) / 9". The expression is written in a cursive style.

Figure II.4: Gina's second expression.

to have seen in the lecture notes by this point<sup>7</sup>. Gina and Benjamin's explicit use of the word "loop", and their concern with how many times something is calculated both seem to support this interpretation. We therefore infer that in their whiteboard inscriptions, they are attempting to represent the code.

Furthermore, in writing her second expression (Figure II.4), Gina indicated explicitly that from her point of view there was a similarity between the variable assignments in the code (the Source Domain) and the mathematical concept of dependent variables (the Target Structure). Because the expression for the value assigned to the variable *diff* (line 4 in Code Sample II.1) contains the value of *x*, Gina realised that these variables were not independent from a mathematical point of view. Furthermore, she refers to the variable assignment of *diff* as an "expression". This seems to suggest that Gina implicitly believed the mathematical definition to be equivalent to the computational assignment, even if she kept using the variable names directly from the code. It should be mentioned, however, that the tutorial worksheet text also used the variable name *x* to describe this number.

At this point, the interviewer intervened, as the students seemed increasingly hesitant and uncertain about what to do next. The interviewer pointed their attention to the nature of their activity, and suggested that it might be taken further in the direction of Math:

**Interviewer** *For if you're not just interpreting the task then you're doing something mathematical, and that is...*

**Gina** *Yes, clearly, I thought it was hard to understand the task, but maybe that's because there are so many steps here.*

The interviewer followed up on this by asking Gina and Benjamin what they were doing on the whiteboard, why they were doing it and what they would use the result for. Gina responded that she supposed it was to visualise for themselves what the problem was.

We find these last two exchanges to be interesting for two reasons. First, they reveal what the students thought they were engaged in (visualising does fit with our label of Code Modelling) and why (there were many steps to the task, and it was hard to understand). Second, it is possible that the students' familiarity with mathematical syntax as opposed to Python syntax could factor into their decision to model the code's behaviour in this way.

At this point, Benjamin seemed to pick up on the interviewer's suggestion that the work on the whiteboard could be interpreted as something more mathematical. While looking at the whiteboard, he interrupted the conversation between Gina and the interviewer, and the following exchange occurred:

**Benjamin** *Because this is very interesting.*

---

<sup>7</sup>See Subsection 1.4.3 in (Mørken, 2017), where two simple sums are represented by loops in one of the first introductions to algorithms. It is possible that this presentation could have made the students think of sums and loops as equivalent in some cases.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

**Gina** *Yes.*

**Benjamin** *Because right if we now insert diff here.*

**Gina** *Mm-hmm* [affirmative].

**Benjamin** [starts writing on whiteboard] *because here it says, right, it says x.*

**Gina** *Yes.*

**Benjamin** *Plus 9.*

**Gina** *Times, yes when we went, let's see.*

**Benjamin** *Equals...*

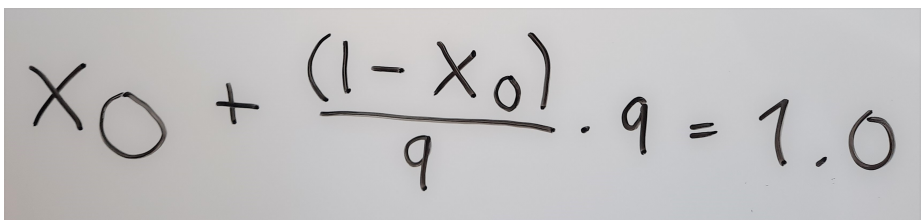


A photograph of a whiteboard showing a handwritten equation:  $X0 + 9 \cdot \text{diff} = 1.0$ . The text is written in black marker on a light gray background.

Figure II.5: Benjamin's first expression.

**Gina** *Yes, that's an equation!*

**Benjamin** *That's an equation, so if we now insert that one* [indicates Gina's second statement in Figure II.4], *then that's basically x divided by 9.*



A photograph of a whiteboard showing a handwritten equation:  $X0 + \frac{(1 - X0)}{9} \cdot 9 = 1.0$ . The text is written in black marker on a light gray background.

Figure II.6: Benjamin's second expression.

**Gina** [eagerly] *Yes!*

We now conjecture that this sequence of statements completes Gina and Benjamin's Code Modelling connection, for three reasons:

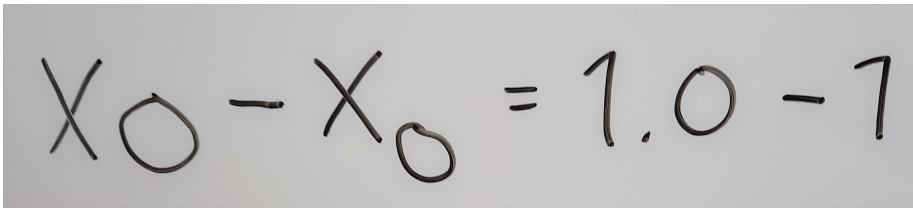
First, what Benjamin did here was to perform a mathematical substitution using Gina's two statements. This appears to be an affordance of interpreting

their whiteboard model as a mathematical construct, because even though Gina noted the dependence of `diff` on `x0` earlier, she did not substitute `diff` for its definition at the time. After all, no such substitution was evident in the code, so for the purpose of "visualising" alone, there was no need to do this. We conjecture that substitution is an affordance of Code Modelling.

Second, we have Gina and Benjamin's exclamations of excitement ("this is very interesting", "yes, wow"). These suggest that in the students' eyes, this was a new possibility that had just surfaced, but had not been evident before. This supports our earlier claim that Gina did not think of the Target Structure as being Math until this point.

Finally, we note a change in Benjamin's use of the Target Syntax in response to Gina's equation statement. In his first statement (Figure II.5), he continued using Gina's syntax with no subscript in  $x$ . In his second (Figure II.6), he adopted the more mathematically conventional  $x_0$ , which he kept using from that point on. The multi-symbol variable `diff` had been substituted away, leaving the constant 1.0 as the only relic left over from the earlier mixed syntax that contained both Math and Code elements. We interpret this as a clear shift in the Target Syntax from a hybrid state towards Math in response to the interviewer's suggestion.

Benjamin proceeded to solve the equation analytically. His work on it is shown in Figure II.7 and Figure II.8. At the end, Gina seemed disappointed in the result:



A photograph of a whiteboard showing a handwritten mathematical expression. The expression is  $X_0 - X_0 = 1.0 - 1$ . The 'X' and '0' are written in a simple, slightly slanted font. The equals sign is centered, and the right side of the equation is written as '1.0 - 1'.

Figure II.7: Benjamin's third expression.

**Gina** *Yes, wow! It cancelled, okay, eh...*

**Benjamin** *So then... but...*

**Gina** *That was a little disappointing, but it's probably that way I...*

**Benjamin** *Eh...*

**Gina** *...be a sensible way to set it up, but...*

We conjecture that Benjamin's attempt to solve the equation is another affordance of Code Modelling. The students made no attempts to solve for anything while thinking of what they were doing as "visualising" the code. It is possible that Benjamin's solution attempt was guided more by the affordances in the Target Structure than by the stated goal of the task: to find out what, if anything, was wrong with the program.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

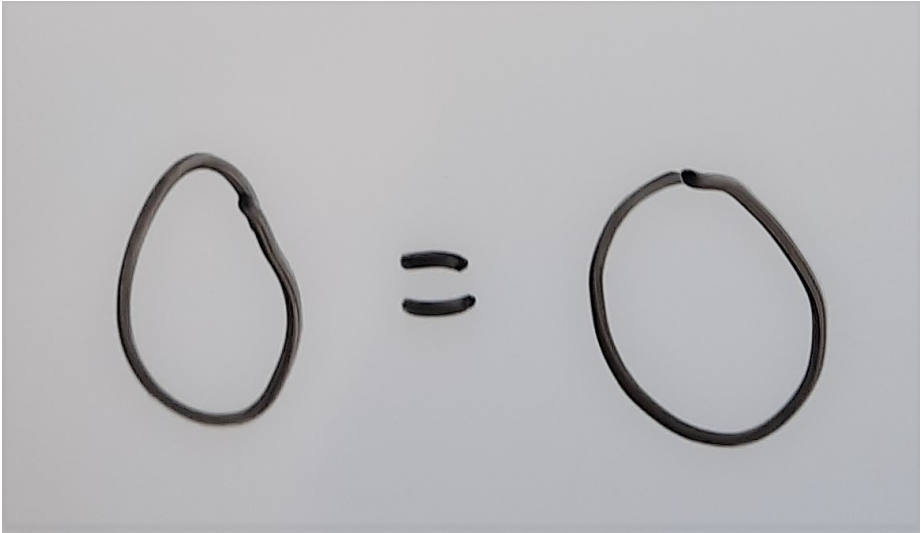


Figure II.8: Benjamin's final expression.

As far as Target Syntax is concerned, in Python,  $1.0 - 1$  in Figure II.7 would be a legal operation, but, under the hood, the integer value 1 would be converted to a decimal number. Hence, the computationally typed result of such a calculation would be 0.0. Benjamin's choice of representing the answer as the integer 0 is further justification for our claim that the Target Syntax is Math at this point, where data types are not considered important. We do not claim that Benjamin consciously left 1.0 as a float value and then made this conversion we described, however. Rather, we believe that Benjamin carried over 1.0 unchanged from Gina's first expression even after the change of Target Syntax to Math (perhaps as an oversight), and then Benjamin ended up converting to mathematical syntax as soon as the value changed.

The final  $0 = 0$  expression (Figure II.8) is interesting for other reasons. From an observer perspective, one may interpret the result  $0 = 0$  as a proof that the equality holds for all values of  $x_0$ . In an equation, a relationship that always holds, such as  $x + x = 2x$ , is reducible to  $0 = 0$  through algebraic manipulation. In contrast, a contradictory statement, such as  $x = x + 1$ , would yield  $0 = 1$  through the same means. In both cases, the variable  $x$  has been eliminated, showing that the truth or falsity of the relation that the equation represents does not depend on the value of that variable.

However, taking an actor-oriented perspective, we infer that Gina's comments reveals that she was not viewing  $0 = 0$  in this way. Rather, she seemed to be almost disappointed in the result and did not see it as helping them progress toward their goals. We speculate that even though this demonstrates constructing a proof as yet another affordance of Code Modelling from our point of view, Gina did not see it as such. Instead, she might have expected to find a particular value



of  $x_0$  that would have made the program behave as intended. In introductory mathematics courses, students are frequently asked to solve for some variable to find a value that satisfies a constraint represented by the equation, and it is not surprising if Gina interpreted the goal of the activity in this way.

With this interpretation in mind, the equation failing to result in a satisfying value might have suggested to her that either such a value does not exist, or the method itself was inadequate to identify it. We conjecture that Gina did expect this method to produce a result in most instances, due to her stated belief in it being a "sensible way to set it up", but that it failed in this particular case. Hence, for Gina, their strategy seemed like it would be useful in general, just not in this particular instance. Benjamin appeared to be of a different mind, however:

**Benjamin** *But, why isn't it working here then, why doesn't that become 1?* [uses the computer to switch from the code editor to the output in the terminal window shown in Figure II.9].

```
(base) C:\Users\Odd Pet
0.26177684777422217
0.3438016424659753
0.4258264371577284
0.5078512318494814
0.5898760265412345
0.6719008212329876
0.7539256159247406
0.8359504106164937
0.9179752053082467
0.9999999999999998
1.082024794691753

(base) C:\Users\Odd Pet
0.17060649344518408
0.2627613275068303
0.3549161615684765
0.4470709956301227
0.5392258296917689
0.6313806637534152
0.7235354978150614
0.8156903318767077
0.907845165938354
1.0000000000000002
```

Figure II.9: The output visible in the terminal window, showing two rounding errors: one that makes the loop take an extra iteration (left), and one that does not (right).

We interpret from this remark that Benjamin was acting as though, from his point of view, the program's method had been proven, although he did not articulate or attempt to defend this belief. If he shared Gina's belief that the program would only work for a certain value of  $x_0$ , or even just that their method was unable to find such a value, we do not think he would act surprised that the program did not produce the expected answer for the two examples in the output. In fact, that would be rather expected behaviour from the program.

We conclude that Benjamin saw the affordance of constructing a proof that Gina did not. The question for him then became why the program failed to reproduce the predicted output. From an observer point of view, a *non-mathematical* reason such as rounding errors would explain the result without invalidating the proof. Because the students did not take such errors into account

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

in their model, their appearance (in theory recognisable in Figure II.9) might have pointed them toward the source of the problem if they had followed this line of reasoning. We do not claim that this insight was available to Benjamin at the time, however.

This change in activity on Benjamin's part signifies a new connection, as their previous Math activity on the whiteboard now became the new Source Domain. The Target Medium became Output, as does the Target Structure (the table of numbers in the terminal window that the students are now considering). We interpret this connection as an example of Mathematical Interpretation, especially on Benjamin's behalf. He seemed to initially trust the mathematical proof but then was surprised the code's inability to accurately reproduce the mathematical result. To the extent that he saw his work as a proof, it is possible that the output made him doubt it.

Gina proceeded to imagine two possibilities: either the program must be told explicitly that the answer needs to be 1 (Source Domain: Code) or an  $x_0$  that satisfies the equation could be found in some other way (Source Domain: Math). We would not consider the former to be a cross-domain connection, but the latter fits the label of Mathematical Interpretation.

Gina's conclusions here point to a possible difference between computing and mathematics that might have been relevant to her: in computing, the answer must be known ahead of time (so one can tell the computer explicitly what it is), whereas in mathematics one can at least attempt to find answers. This might help explain why they needed to go fully into the mathematical syntax of Code Modelling before they were able to construct a proof of concept. It is possible that from these novice programmers' point of view, only mathematics can produce answers for the computer to make use of, whereas the computer can only work with information that is known explicitly in advance.

One possible consequence of this mindset is the notion of the program only working perfectly for a particular  $x_0$ , which had not been evident before this exchange. The idea seemed to resonate strongly enough with the students that also Benjamin appeared to believe in it himself later on, at the expense of his initial interpretation that was more in line with the observer point of view. This belief effectively prevented Gina and Benjamin from focusing on the important difference between mathematical theory and computational practice until the interviewer intervened.

To summarise, in this case, we identified that the students made cross-domain connections, and we explored what these connections afforded. First, Gina's uncertainty about how the code worked led her to transition into modelling it using the more familiar (to her) syntax of mathematics. This afforded their focus to shift from the code itself (and its output) to what it represented (the mathematical way these results were calculated).

Second, the interviewer's observation that the model of the program could in fact be interpreted as something inherently mathematical seemed to cause Benjamin to ignore what the statements represented and instead take them as mathematical givens. This afforded the use of the students' repertoire of mathematical tools, resulting in a proof of concept for the program. We interpret

this as the completion of the connection that began with Gina's first attempt on the whiteboard.

Finally, Benjamin's apparent acceptance of the proof led him to shift his focus to the code's output, to make sense of the dissimilarity between the mathematical and the computational result. This evaluation could potentially have afforded a useful investigation into the differences between the mathematical statements and the code that they modelled if not for Gina's rejection of  $0 = 0$  as a useful result. Instead, Benjamin's focus on the output allowed Gina to express her own ideas on how to achieve the program's stated purpose: either instruct the program in clear terms what the answer should be, or somehow identify a value of  $x_0$  that would make this happen by mathematical means. We believe that these two ways of ensuring the desired result are treated as equally viable from the students' point of view is a feature of their integrated approach to working with computing in mathematics.

## II.4.2 Case B: Rita and Lena (Tutorial 2, 2020)

Our second case is about Rita and Lena's work on Tutorial 2. This case highlights how students can flexibly go back and forth between computing and mathematics to suit their immediate needs.

### II.4.2.1 Tutorial 2: Logarithms and Taylor Expansions

The second tutorial (described in greater detail in Paper III) asked students to write their own function for calculating logarithms using a Taylor expansion. The first part of Tutorial 2 gave students functioning code to calculate the Taylor expansion of the natural logarithm. Then, it asked them to implement a function to calculate the remainder for their choice of  $a$ , the point chosen as the basis for the Taylor expansion, and  $n$ , its number of terms. After having implemented a function that calculated the remainder, the students were given the opportunity to experiment with the two key parameters  $a$  and  $n$ , and observe the results.

The tutorial's second part asked the students to exploit the binary representation of floating-point numbers in the computer to relax the accuracy requirements on using Taylor expansions to calculate logarithms. This is of particular interest because it mirrors how algorithms that calculate logarithms are made, even though approximations that require fewer terms than Taylor expansions are often used in practice.

The basic idea is as follows: Taylor expansions are usually intractable to use directly because they require a very large number of terms to reach the desired accuracy far from the fixed point  $a$  one chooses as a basis to construct the expansion. However, by exploiting the way floating-point numbers are represented in the computer's binary memory, one may map the half-open interval of all positive real numbers to the closed interval of numbers between 0.5 and 1.

This transformation is more impactful than it might appear at first glance. Achieving the desired accuracy for this small interval is vastly simpler than doing

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

so for all positive numbers. This means that we can indeed use the computer's proficiency for working with basic calculations (such as polynomials) to represent more complex functions such as the logarithm.

At the end of the tutorial, the students were asked to implement their own log function in this way. With this done, the final task was for the students to make use of their earlier work on the remainder to pick optimal parameters for the Taylor expansion. The goal was that their log function would become machine-accurate on the one hand and as fast and efficient as possible on the other.

### II.4.2.2 Case B: Interview and Analysis

In this interview, the students adhered to social distancing guidelines because of COVID-19 safeguards: Rita wrote on the whiteboard and Lena did all the typing on the computer keyboard. Lena had previously taken IT classes<sup>8</sup> in high school but mentioned that the class focused mostly on creating webpages and working with databases. Rita had no prior programming experience, but she displayed familiarity and skill with many mathematical methods throughout the interview. Rita and Lena had already been interviewed together for an updated version of Tutorial 1 - this was their second interview together.

In this case, we use timestamps in the interview excerpts to indicate where they appear out of chronological order. This happens where we needed additional data from the follow-up questions at the end of the interview to make our point.

In this segment, Rita and Lena were in the final part of the tutorial. They had correctly calculated the remainder and completed writing their own logarithm function as described in the previous subsection. Their next task was to use the remainder to check that their choices of  $a$  and  $n$  had achieved the desired accuracy in the interval  $0.5 \leq x < 1$ . As it turned out, their initial choices led to a much higher accuracy than the tutorial required (see Figure II.10), which prompted the following challenge from the interviewer:

**Interviewer** [1:06:19] *You've seen that it's faster<sup>9</sup> when we have fewer terms.*

**Lena** *Yes.*

**Interviewer** *How fast can we get it to go? With that  $\ln^2(a)$ ? How few terms can we get away with?*

**Rita** *Ehm... I guess one has to... One could calculate it. Or, sort of. We could either just try with a lot of different  $n$ 's. [laughs]*

**Lena** *And stopwatch and just see how fast it goes? [laughs]*

---

<sup>8</sup>These Norwegian IT classes referenced here predate the recent initiative to introduce programming into the high school curriculum. While some programming concepts are covered, these courses were very general, and covered everything from databases to making web pages. Lena herself expressed that she did not feel those classes had prepared her very well for tasks such as the one in this tutorial.

<sup>9</sup>In this context, "fast" refers to the running time of the program.

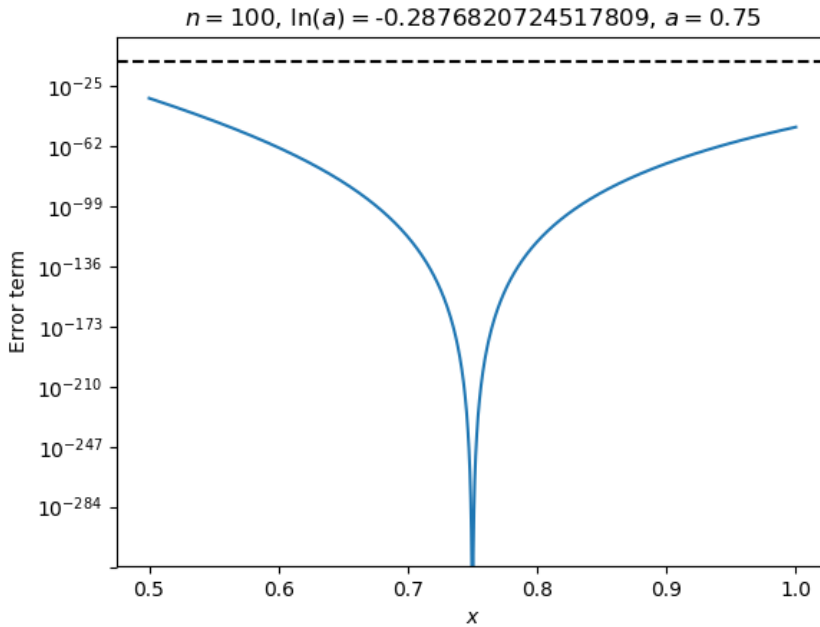


Figure II.10: The students' initial results for  $a = 0.75$  and  $n = 100$  led to a much lower absolute value for the remainder than the stated goal of  $10^{-10}$  in the entire interval (the dashed line in the plot).

**Rita** *And just take the smallest  $n$ , if not then you can sort of set that, if you set the remainder as...  $10^{-10}$ , then.*

**Lena** *Yeah. Can we manage that?*

**Rita** *And... Sort of, or if you have to calculate<sup>10</sup> it. I don't know.*

We interpret this exchange as Rita trying to decide between two options in response to the interviewer's suggestion: some form of trial and error or a more rigorous mathematical calculation. She proceeded to ask Lena to open the document where the formula for the remainder was displayed. Rita then asked the interviewer whether they should try to solve that expression for  $n$  or use trial and error instead. The interviewer conceded that the latter was the original intent, because there would have been a lot of  $x$  values to check for otherwise.

**Rita** *Oh yeah, no, but I thought that we just tested for 0.5 and 1 because that's the worst case.*

<sup>10</sup>In Norwegian, the implication was that they would do a calculation by hand; this distinction was obfuscated in translation.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

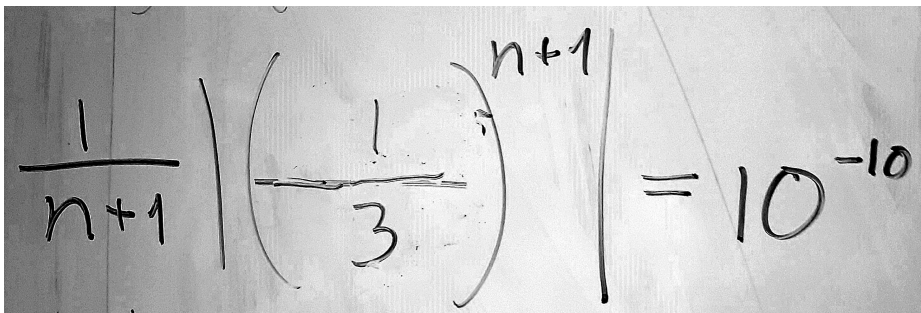
---

**Interviewer** *You wanted to test the endpoints where, sort of, it's the worst case?*

**Rita** *Yeah.*

**Interviewer** *That sounds like a really good idea to me.*

Rita did express some uncertainty about how to do it but proceeded to write down an equation where the remainder was equal to  $10^{-10}$ , the limit for an acceptable remainder according to the tutorial. With Lena's help, she substituted the required values for  $x = 1$  into the remainder formula, resulting in an equation with  $n$  as the only unknown (see Figure II.11).



The image shows a whiteboard with a handwritten mathematical equation. The equation is  $\frac{1}{n+1} \left( \frac{1}{3} \right)^{n+1} = 10^{-10}$ . The terms are written in black marker on a light-colored surface. The fraction  $\frac{1}{n+1}$  is on the left, followed by a multiplication sign and a large parenthesis containing  $\frac{1}{3}$  and an exponent  $n+1$ . This is followed by an equals sign and  $10^{-10}$ .

Figure II.11: Rita's equation for the remainder at  $x = 1$  after substituting in the known quantities. The right-hand side is the upper limit of an acceptable remainder, according to the tutorial.

This activity fits the Output Modelling label. Rita going to the whiteboard to perform the calculation and stating that she wanted to do so analytically show us that the Target Medium and Target Structure are both Math. Figure II.11 shows us that the Target Syntax is also mathematical, using conventional symbols that are not used in Python and single-letter variable names. We infer that the Source Domain is Output, as the interviewer's challenge that the students responded to points back to the plot in Figure II.10. We also note that Rita consulted a mathematical resource (the formula sheet that accompanied the tutorial) instead of looking at the code they had implemented earlier.

We can also find supporting evidence in the follow-up questions at the end of the interview, when Rita and Lena talk about why they did the whiteboard calculations:

**Lena [1:48:33]** *I think it was to get an overview, sort of. That it's easier to get it written down in a little larger...*

**Rita** *Yeah, I'm stronger... or I'm better at math than I am at programming.*

**Lena** *Yeah.*

**Rita** *So I often feel that if it's something that's calculable<sup>11</sup>, I feel it becomes more correct when I do it mathematically.*

**Interviewer** *Or clearer, perhaps?*

**Rita** *Yeah, I kind of like having an expression for things.*

From the student point of view, it appears that this Output Modelling afforded clarity and confidence in the result. It also afforded the use of familiar (to the students) mathematical tools like the substitution and simplification Rita made use of. This mirrors Gina and Benjamin's Code Modelling in Case A.

$$\frac{1}{10^{10}} = (n+1) \cdot 3^{n+1}$$

$$\ln(10^{10}) = \ln(n+1) + ($$

Figure II.12: The endpoint of Rita's first whiteboard calculation.

The students then spent some time manipulating this expression analytically on the whiteboard (see Figure II.12), but eventually concluded that it would be difficult for them to solve. It turns out that their hunch was correct, as the expert solution involves a function called the *product log function* that is only implicitly defined as the solution to equations on this form ("Wolfram Alpha", n.d.). Importantly, then, they reached a point here where they did not have a clear way to proceed from a strictly analytical perspective. Instead, they found another way forward:

**Rita [1:15:34]**  $\ln(10^{10})$  equals  $\ln \dots$  *But we don't get an expression for  $n$  in this case.*

**Lena** *No. [inaudible]*

**Interviewer** *What's the problem here? Or what were you about to say, Lena?*

**Lena** *Mmm. I was about to ask whether it'd be easier to write that formula here in Python or something. Whether we're able to calculate it in that case.*

<sup>11</sup>She means analytically (see footnote 17).

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

Rita and Lena clearly stated that the operations they saw as available to them would only make them come full circle, and to progress they would have to take another approach. We note with interest that Lena suggested Python without prompting on the interviewer's part. In the follow-up question part of the interview, she elaborated on this transition back to computing:

**Interviewer [1:47:08]** *But how did you come up with "hey, we can do this in Python", Lena? Because that was also sort of a breakthrough idea that helped us further.*

**Lena** *Yeah. No, really, the thing is that when there are very complicated expressions, or like when we have something raised to the power of 1000 or something like that.*

**Interviewer** *Mm-hmm [affirmative].*

**Lena** *Then you do quickly think that it can't be done by hand. But that it's quite easy to type. And then you get an answer. So then you save some time.*

She also commented on what this transition afforded her:

**Interviewer** *So, but this way of working, does it enable us to do things we couldn't otherwise? If we should just have done math or just...*

**Lena** *Absolutely.*

**Rita** *Mhm. We wouldn't have been able to find that expression for  $n$ .*

**Lena** *No, not without using programming to do it.*

Lena proceeded to open a new Python file in the code editor and translated the equation on the whiteboard to the form in Code Sample II.2, which contained no fractions or logarithms. We interpret this as Lena considering such a form to be the simplest one to solve, because this was not the form that they had written on the whiteboard in Figure II.12. In this, Lena was in alignment with the expert perspective, except for getting Python's assignment operator = confused with == that tests for equality.

```
(n+1)*3**(n+1) = 10**10
```

Code Sample II.2: Lena's translation of the equation in Figure II.12 to the code editor (for  $x = 1$ ).

The students were unsure how to progress from there, however:

**Rita** *How do you, like, get that to solve for  $n$ ?*

**Lena** *Yeah. Are there some functions in Python? "Solve equation" or something like that?*

**Rita** *Maybe. Try some thing or other. Just write something like "from math import star", it's guaranteed to be a, sort of, math module.*



```
from math import *
math.solve((n+1)*3**(n+1) = 10**10)
```

Code Sample II.3: Lena's first attempt at a Python solution.

Lena made an attempt at such a solution (see Code Sample II.3), but the interviewer discouraged this approach due to time constraints and the requirement of using Python libraries unfamiliar to the students like `sympy`.

We classify this connection as Math Implementation. The Source Domain is Math, through Rita's earlier whiteboard work. The Target Medium and Syntax are Code, as the focus was on Lena's work on the computer at this point. As for the Target Structure, we claim that this represents the first stage in a two-step connection, much like Gina and Benjamin in Case A, with two important differences: (a) the Target Syntax is not a hybrid state containing elements of both Math and Code<sup>12</sup>, and (b), Lena's work is a computational representation of a mathematical entity, which is the opposite of what Gina and Benjamin made. Like in Case A, this would soon change as the interviewer intervened.

First, however, we note that the way the students first attempted to implement the problem was still very closely linked to their earlier analytical whiteboard work. They initially wanted the computer to do the exact same thing they had attempted themselves by hand. When that did not work, they did not seem to see how to connect the equation to familiar Python concepts on their own, perhaps due to their still interpreting it as something mathematical. Wanting to remain in the Python environment, the interviewer offered an alternative way to accomplish the task using Python:

**Interviewer** *We could just try different  $n$  values and see whether it becomes  $10^{10}$ , though.*

**Lena** *Yeah.*

**Interviewer** *In this one. But it could also be that there is a way to have Python try... a lot of different  $n$  values for us. So that we don't have to try each and every...*

**Rita** *Oh, yes.*

**Lena** *For loop?*

**Rita** *If we write, like, while or for, yeah.*

**Lena** *Yeah. That's not such a bad idea.*

<sup>12</sup>In theory, one could object that the use of single-letter variable names points us toward Math, but in Code these are perfectly legal. Furthermore, while shortening variable names so that they are not interpreted as the product of many variables ( $cat \neq c \cdot a \cdot t$ ) makes sense when representing Code using Math, as Gina and Benjamin did, *lengthening* variable names makes little sense from an expert perspective, as there is no such ambiguity in Lena's resulting expression (Code Sample II.2).

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

The interviewer's first idea ("try many different values") was enthusiastically received by the students, who immediately related this intentionally vague statement to the computational concept of loops. The interviewer's further suggestion that they use a while loop that would stop when the answer is good enough was immediately connected to and expressed as an inequality by Rita, which Lena then leapt at the chance to implement. In other words, they articulated a connection between the mathematical educated guess (picking a possible solution that seems reasonable and checking if that satisfies the inequality) and the computational iteration (trying out every possibility in ascending order until the inequality is satisfied).

This connection completes the transition into Math Implementation. The change from earlier is that the Target Structure seems to have changed from Math to Code at this point. This is based on the students implementing the inequality as a test condition that yields True or False, which is structurally different from simply enclosing an equation within `math.solve()` as they did earlier (in Figure 15).

The affordance of this connection is a better understanding of the process of determining the optimal value of  $n$ , which now features in a computational role as the loop variable in addition to being the mathematical unknown that is solved for. The additional affordance to leverage their coding experience without resorting to symbolic algebra libraries was not lost on the students. They explicitly rejected the idea of using the `sympy` library for the task at hand, given their lack of familiarity with it.

Instead, they started setting up two possible loop structures (*while* and *for*) and initially tried to combine them (see Code Sample II.4), but eventually decided that the *for* loop was unnecessary and with some input from the interviewer on correct syntax arrived at a functioning *while* loop (Code Sample II.5).

```
for n in range(100):
    while ((n+1)*3**(n+1) < 10**10):
```

Code Sample II.4: Competing loop possibilities, combined.

```
n = 1
while ((n+1)*3**(n+1) < 10**10):
    print(n)
    n += 1
```

Code Sample II.5: Lena's final version of the loop.

At this point, the students tried running the code, resulting in the output seen in Figure II.13. The inequality was satisfied as long as the remainder was too large - in this case, the loop would continue. It stopped at the point where the remainder was acceptably small, hence the final printed value represented the minimum number of terms that produced an acceptable result:

Lena proceeded to modify the code the students used earlier, reducing the number of terms from 100 to 17, while keeping the fixed point  $a$  of the Taylor expansion unchanged. Unfortunately, attempting to use the newfound minimal

```

14
15
16
17
(base) D:\h20_v21\Data\T2-C>

```

Figure II.13: Output of the students' program for  $x = 1$ , with  $n = 17$  as the number of terms that keeps the remainder sufficiently small.

value  $n = 17$  did not result in the desired accuracy in the *entire* interval between 0.5 and 1, as it proved to be insufficient for low values near  $x = 0.5$  (see Figure II.14). The students' first reaction to the result suggested scepticism, as they remained completely stationary upon seeing the plot, and a significant pause followed before they spoke in a hesitant tone of voice. The students did not immediately buy into the interviewer's interpretation of the result as a partial success:

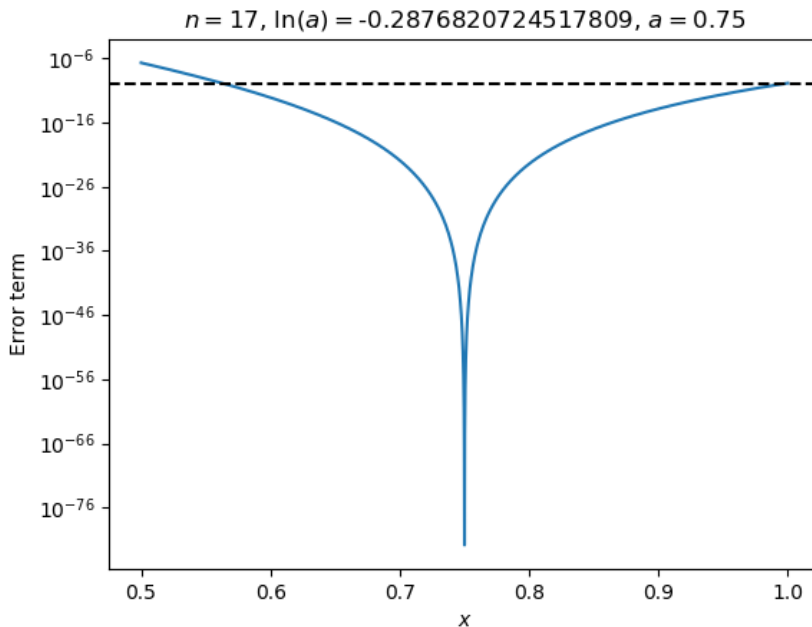


Figure II.14: Plot of the remainder with  $n = 17$ .

**Rita** *Ehhh, yes. Right.*

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

**Interviewer** *I thought that worked well, actually.*

**Rita** *But, it's not below  $10^{-10}$  though.*

**Lena** [*inaudible*] *Yeah. How do you think that works well?*

**Interviewer** *Because I'm looking at the point  $1.x = 1$ . And there's it's sort of a hair's breadth below.*

**Lena** *Yeah.*

**Rita** *Oh yeah, that's what we calculated, yes. Yeah, because then we should do that for both sides, then. But that... yeah. [The others voice agreement.] Because that did work well. And then you just take the largest  $n$  value of the two endpoints.*

Here, Rita made another cross-domain connection between the output (plot) and her earlier work on the whiteboard: one needs to choose the point on the curve that maximises the remainder to say that the remainder is acceptably small in the entire interval. We label this connection Mathematical Interpretation, as the Target Medium, Structure and Syntax were all the Output that the students attended to, while the Source Domain was Math (pointing back to Rita's original choice  $x = 1$  when setting up the original equation). This connection afforded validation of the result and making a better choice for the next attempt.

We conjecture that a fourth connection was made shortly thereafter when Rita proceeded to repeat the calculation for  $x = 0.5$  and the interviewer pointed out that the parameter  $\xi = \min^?(ax)$  in the remainder formula would also change because of the change in  $x$  to a value smaller than  $a$ . Rita then immediately reasoned that these changes would in fact explain why one endpoint produced a greater remainder than the other:

**Rita** *Yeah, that's it, because then you're dividing by a smaller number, and then it does become larger...*

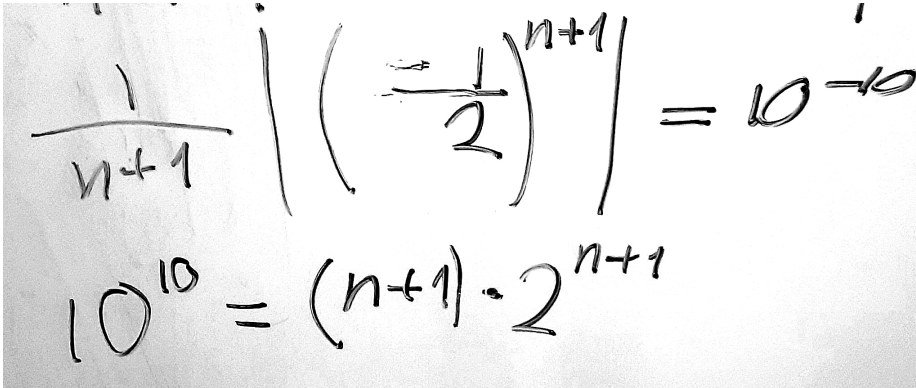
This connection is another example of Output Modelling. We claim that the Source Domain Rita is referring to would be Output (the plot in Figure II.14), where the fact that  $x = 0.5$  had a greater remainder was evident. The Target properties were all Math, as Rita was working with the new equation at the whiteboard at this point, repeating the work she did earlier where we assigned the same label. This connection then afforded explaining why changing the value of a variable had a particular effect on the output.

She then completed the calculation with these new values on the whiteboard and arrived at the result in Figure II.15, which Lena then used as the basis for modifying their Python program. She then used the output (Figure II.16) as an input parameter to the remainder function, resulting in the plot of Figure II.17.

Upon seeing the result in Figure II.17, both the interviewer and the students expressed satisfaction with the result:

**Interviewer** *Wow.*

**Rita** *Shit, we're smart.*



$$\frac{1}{n+1} \left| \left( -\frac{1}{2} \right)^{n+1} \right| = 10^{-10}$$

$$10^{10} = (n+1) \cdot 2^{n+1}$$

Figure II.15: Rita's calculation for  $x = 0.5$ . Note the similarities with the expressions in Figure II.11 (top) and Code Sample II.2 (bottom).

```

24
25
26
27
(base) D:\h20_v21\Data\T2-C>

```

Figure II.16: Output of the students' code for  $x = 0.5$ .

**Lena** *[laughs]* Yeah.

As a final test, Rita and Lena used  $a = 0.75$  and  $n = 27$  as input to their logarithm function. A comparison between the commonly used `numpy` library's `log` function and the students' own results can be found in Figure II.18.

**Lena** *It's quite a lot of identical digits.*

**Rita** *Wow, now it was really nice.*

**Lena** *Oh, my god. Relative error 0.0.*

To sum things up, the students were faced with the dilemma of how many terms to include in the Taylor expansion. They connected that question to a mathematical equation. When this failed to provide an analytical answer, they further connected the equation<sup>13</sup> to a numerical trial-and-error approach using

<sup>13</sup>Which by then had been transformed into a form that made it simpler to use numerically, with no need for a function to calculate the absolute value, for instance. From an expert perspective, one could argue that the students could have done this without the mathematical work, but the fact that the students used the results of their work indicates that it had value to them.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

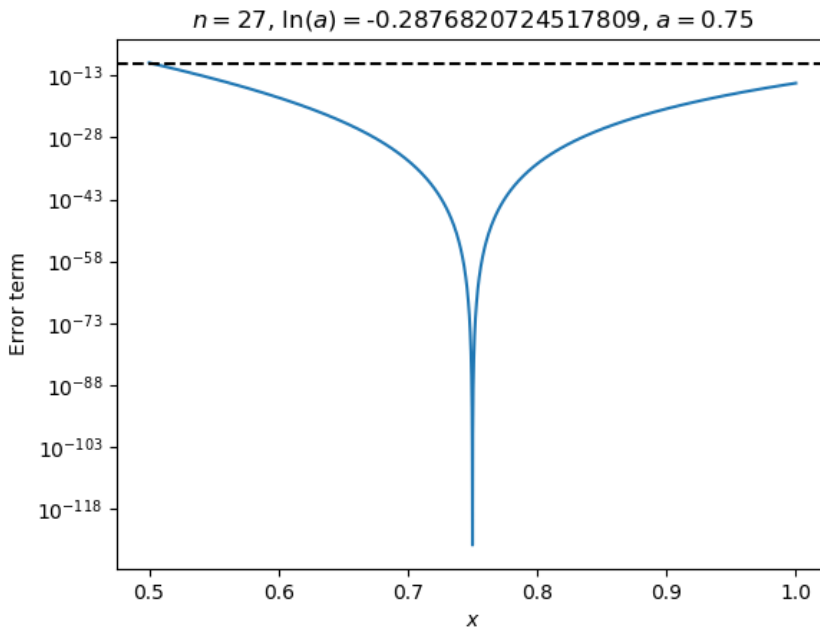


Figure II.17: Result of using the number of terms in Figure II.16.

```

x = 10
---
log:          2.302585092994046
taylorlog:    2.302585092994046
relative error: 0.0

x = 0.001
---
log:          -6.907755278982137
taylorlog:    -6.907755278982137
relative error: 0.0

x = 0.1
---
log:          -2.3025850929940455
taylorlog:    -2.3025850929940455
relative error: 0.0

x = 1000
---
log:          6.907755278982137
taylorlog:    6.907755278982137
relative error: 0.0

ln(a) = -0.2876820724517809
a      = 0.75
n      = 27
    
```

Figure II.18: Tests of the students' own log function.

a loop. Starting with Output, the students went from there to Math and then to Code. The computer then closed the circle by taking the Code and producing new Output.

The students then made further connections by using mathematics to explain what had happened and decide what to do next when the plot surprised them. As they re-did the whiteboard calculations, they referred to the plot and explained what happened to the remainder using mathematics. This set of connections thus took them from Math to Output and back to Math again.

What these connections have in common is that each of them afforded a new perspective that provided deeper insights into the problem. The guessing game to find the ideal value for the parameter  $n$  transformed into a rigorous mathematical solution with the potential for greater explanatory power. The realisation that the equation was hard to solve analytically transformed into a systematic way to find a numerical solution. They were now able to explain *why*  $x = 0.5$  was the better choice, not just that it was better than  $x = 1$ . And finally, they were able to justify their choices of parameters as ideal with regards to calculating the logarithm with as few terms as possible.

### II.4.3 Case C: Lydia, Martin, and Roger (Tutorial 3, 2019)

In our final case, we present work by Lydia, Martin, and Roger on Tutorial 3. This case highlights how a task that asked the students to implement mathematics on the computer facilitated fruitful mathematical discussion.

#### II.4.3.1 Tutorial 3: Numerical Integration

The final tutorial had students numerically integrate a familiar function whose integral value is well known but impossible to calculate analytically: the standard normal distribution. The students were tasked with implementing a function that calculated one term in the Riemann sum using the midpoint method (they were given code that would then use their function to calculate the entire integral).

It is worth noting that the midpoint method with step length  $h$  they were asked to use,

$$\int_a^b f(x) dx = \sum_{x=a}^b f\left(x + \frac{h}{2}\right) \cdot h$$

can be interpreted as a special case of the more general Euler's midpoint method for differential equations (Mørken, 2017),

$$\begin{aligned} x' &= f(t, x) \\ x_{k+\frac{1}{2}} &= x_k + \frac{h}{2} f(t_k, x_k) \\ x_{k+1} &= x_k + h \cdot f\left(t_k + \frac{h}{2}, x_{k+\frac{1}{2}}\right) \end{aligned}$$

Note that in the latter case the symbol  $x$  has taken on a different meaning (a function) than in the former (where it is the independent variable). Expressing

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

the integral midpoint method with differential equation notation would look like

$$\int_a^b f(t, x) dt = x_0 + \sum_{k=1}^n x_k = \sum_{t=a}^b f\left(t + \frac{h}{2}\right) \cdot h$$

The reason for this is that in the special case of the integral,  $f(t, x)$  has no dependence on  $x$  and the boundary condition is  $x_0 = x(a) = 0$ . It is perhaps not surprising, then, that the students in Case C would confuse the two and treat these as two unrelated methods. They really do look different from one context to the next.

The central question in this tutorial was how many steps (terms in the Riemann sum) would be needed to balance accuracy with efficiency. After being encouraged to try out some values and note the results, the students were asked to plot the relative error as a function of the number of steps. The goal was that they should discover a sweet spot, where the mathematical error is as small as possible, and the rounding errors (that increase with more terms) also having minimal effect.

### II.4.3.2 Case C: Interview and Analysis

Lydia was verbally active and asked several questions throughout the interview. She often asked the group to pause their current activity to discuss the underlying concepts. Martin stated that he had previous experience using Python from work, and Lydia and Roger often deferred to him in times of discussion. Martin also typically initiated the writing or typing on the keyboard. Roger's interactions with the group mostly concerned completing the given tasks. The three students were used to working together from the group sessions in class.

Lydia, Martin, and Roger had just been asked to implement the midpoint method for numerical integration in Python (not the complete integral, but a function that returned a single term in the Riemann sum). Lydia's first reaction was to mention that this sounded like Euler's method.

**Martin** *OK, so... should we...*

**Lydia** *... run it by hand first?*

**Martin** *Or, I just thought about drawing it.*

**Lydia** *Yes*

**Martin** *So, you have a function...* [begins drawing on his worksheet]

We interpret this exchange as another example of Code Modelling. This drawing, as seen in Figure II.19, was done on the tutorial worksheets that we collected, hence the Target Medium is Math, as is the Target Syntax (variable and function notation) and Target Structure (a plot of a function with boxes representing steps in the integration). The Source Domain is the Code, namely the function they were asked to design and implement. They had opened the code editor before Martin started drawing.



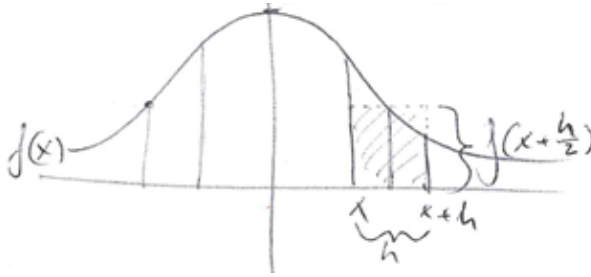


Figure II.19: The drawing from Martin's worksheet. This version of the drawing is likely more complete than it is when it is first referenced in the transcript (Martin returns to this drawing several times during this episode).

In contrast to Case A, the students were not deciphering a piece of code they had been given but were instead asked to write and implement the code themselves. They also used a mathematical drawing rather than setting up an equation to solve. Nonetheless, Lydia's comment about "running it by hand" reminds us of Gina's "by hand" comment in Case A. Unlike that comment, however, Lydia explicitly blended a computational construct ("running") with a mathematical one ("by hand"). The possibility that something written by hand could be *run* supports our interpretation that Lydia was aiming at Code Modelling much like it played out in Gina and Benjamin's case.

After completing the first version of the drawing, computational ideas kept emerging in the students' conversation:

**Roger** *What was  $n$ ?*

**Martin**  *$n$  is the number of points we've been given...*

**Roger** *Yes, OK.*

**Martin** *... or the size of the precision.*

**Martin** *Yeah, yeah.*

**Lydia** *Now you're multiplying [inaudible] along.*

**Martin** *Ehm, does anyone remember the midpoint method?*

**Lydia** *Yes, first we must go a half-step [points to Martin's drawing].*

**Martin** *Yes [keeps drawing].*

**Lydia** *Ehm... and then we have to store that variable.*

**Martin** *Mm-hmm [affirmative].*

In this conversation, we note that the students used language that referenced not only computational concerns, but also specifics of the implementation. Computational elements were re-introduced when Roger asked about a computational variable ( $n$ ) that was not present in the drawing. Moreover,

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

Lydia's explanation of the midpoint method explicitly referenced storing a result in a variable, which would be an unconventional way of putting it in traditional mathematics.

It is also possible that this comment points to what Lydia meant earlier by "running the code by hand": doing a calculation on paper while at the same time imagining what it would take to implement it on the computer and considering the process of that implementation. We find this to be consistent with our idea of Code Modelling.

What this connection appeared to afford the students was, much like in Case A, a visualisation of the code and method they were asked to implement. Unlike Gina and Benjamin, these students did not apply mathematical tools like substitution and equation solving. Instead, they used Martin's drawing as the basis for a mathematical discussion about the midpoint method and its relation to other concepts, and this discussion continued throughout the interview.

The discussion of how to implement the midpoint method initially lasted until Martin recognised that they had something calculable, which we identify as Math Implementation:

**Martin** [looking down at his drawing] *f of x plus h halves.*

**Roger** *Yes.*

**Lydia** *h divided by two, yes.*

**Martin** *...yes. We are able to calculate that.*

At this point, Martin began modifying the Python code, as shown in Code Sample II.6:

```
def midpoint(x,h):  
    return h * f(x+h/2)
```

Code Sample II.6: Martin's implementation of the midpoint function.

Martin thus produced a computational function to represent a mathematical function<sup>14</sup>, and chose variable names so that the computational variables represented their mathematical counterparts. This could suggest that the Target Syntax is Math, but the variable names  $x$  and  $h$  were given in the tutorial itself and lengthening them would be uncommon (see footnote in Case B). Furthermore, failure to make the Target Syntax Code would likely have led to error messages.

The Target Medium is Code, as is the Target Structure, because Martin was writing a function in the code editor<sup>15</sup>. His utterances as he wrote the

---

<sup>14</sup>This distinction is perhaps not obvious. While a mathematical function is a one-to-one mapping between two sets of elements (typically numbers), computational functions can perform tasks that have little to do with the mapping between input and output, such as writing to the screen or saving to a file. It is therefore prudent to regard mathematical functions that can be represented computationally as a subset of all computational functions.

<sup>15</sup>It would be uncommon, but possible, to use the code editor as a sketchpad to solve an equation analytically. That would make the Target Structure Math, as opposed to what we describe here.

function strongly suggested that the Source Domain is Math, as there would be no differentiated function available to him in the code. His claim that "we are able to calculate that" also connects the code he subsequently wrote to the earlier mathematical midpoint method discussion between the students, at which point he was focused on his drawing.

This suffices for us to interpret this connection as Math Implementation, which suggests to us that the variables were intended to be Code representations of the original Math variables. What this connection afforded was an implementation of the method they had been recalling and describing. We note that while doing Code Modelling the students were attending to the particulars of the implementation, but they ended up using none of the particulars they discussed (the number of terms and storing the result of a half-step in a variable). Even so, we claim that the Code Modelling laid the foundation for the method discussion that was later transferred into the Math Implementation.

The students then evaluated the work to see if what Martin had done made sense. Martin asked the interviewer (who had informed the students that he functioned as a TA in this setting) a question:

**Martin** [to interviewer] *Is it the same as the Euler midpoint method, where he takes a half step to find one point and its derivative and then he takes the derivative...*

**Lydia** [draws parallel, slanted lines in the air with her hand] *and then he makes another one below that is parallel with the other line [laughs] where everyone reacted to [the instructor] drawing it very badly... heh.*

To see what the other students made of the question, the interviewer withheld his answer until the discussion came to a stop, which did not occur. Instead, the students confirmed that their implementation gave the result that they were supposed to get (as indicated in the tutorial).

At this point Martin and Lydia both seemed to be seeking to reconcile the method they just wrote with what they learned in the recent lectures on Euler's method. Roger seemed less concerned as long as what they did both worked and made sense in itself. Martin then expressed his belief in there being two different midpoint methods, and Lydia noted that the one they had used seemed much easier to use than the method she remembered from the lectures. Finally, Martin made a connection that allowed them to distinguish between the two methods:

**Martin** *Now I have to think a little... [pause] No, that one's for differentiation...*

**Roger** *Yes... because then you take... then you take the tangents, don't you?*

**Lydia** *Fo... [inaudible]*

**Martin** *Yeah.*

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

**Roger** *Yeah, in each point.*

**Martin** *Because if you're coming from here [indicates the drawing in Figure II.19] then you should remove the next point. . . No, that's for differential equations, of course!*

**Lydia** *There it is, there it is.*

**Martin** *There it was.*

**Lydia** *Now I follow you, there was something that really didn't fit there [laughs].*

This represents something of a breakthrough for the three of them. We claim that the realisation that there were two different midpoint methods (from their point of view) and the subsequent categorisation of their uses both sprung from the computational task of implementing the method for numerical integration. The same can be said for Lydia's successful attempt to categorise the method they *had* used, which she recognised as Riemann sums.

To summarise his episode, this group of students went back and forth between Math and Code as they designed their function, and only subsequently made use of the Output to determine its correctness. Finally, they took a step back from the code to a mathematical discussion which helped them see similarities between their work and earlier lectures.

### II.5 Discussion and Conclusions

In these three cases, we have seen students make connections between mathematical and computational ideas as they worked with tasks that integrated knowledge from these two domains. Each case demonstrates a distinct way that students connect these ideas, often in ways that they had not done earlier in the interviews, or in ways that were not anticipated by the tutorial designs. Our cases also represent mathematical contexts in which mathematics and computing were integrated, and these exemplify the kinds of situations in which such integration may be leveraged and explored.

#### II.5.1 Summary of Cases

This subsection provides summaries of the connection types that surfaced in each of the three cases. We will identify four unique *patterns*, defined as certain connection types appearing together, that the cases exemplify. These patterns are summarised in Table II.3. The affordances we link to each pattern in that table are taken from our analysis of the connections as they appeared in the analysis in Section II.4.

In Case A, we saw Gina and Benjamin use mathematics to model the program we gave them, in order to grasp how it was supposed to work. After having translated the program to mathematical symbol language, they were able to take those expressions and treat them like mathematical quantities such as equations.

We posit that this kind of algebraic manipulation would not have been so easy for them to do in a strictly computational setting, as it would require knowledge of `sympy` or similar Python libraries.

Through manipulating and reasoning about these mathematical equations, the students in Case A were able to construct a formal proof of the program's intended function. If both had recognised this proof for what it was, we suspect the resulting comparison between the proof and the output of the program would have provided them with a rich opportunity to identify the problem with the program and to discuss numerical problems versus mathematical problems in general.

This implies that making these kinds of cross-domain connections can lead students to gain profound insights about Code through thinking of it as Math, especially when it comes to information that is *implicitly* present in the code and requires mathematical thinking to unpack. This notion is in stark contrast to Gina's belief that the program must be told everything in clear terms. In any case, their Code Modelling supported the students' work, because familiar algebraic manipulation highlighted a feature of the problem that they (correctly) perceived as relevant.

Gina and Benjamin followed up this work by going into Mathematical Interpretation of the Output, allowing them to link Code and Output together through Math instead of using the program as a black box to produce the output for them. We have named this pattern *Replicating Program* and depicted it in Figure II.20. Note that this only characterises these students in *part* of the interview we used for Case A - in other parts of the interview the same students' connections may be described differently (the same applies to Case B and Case C).

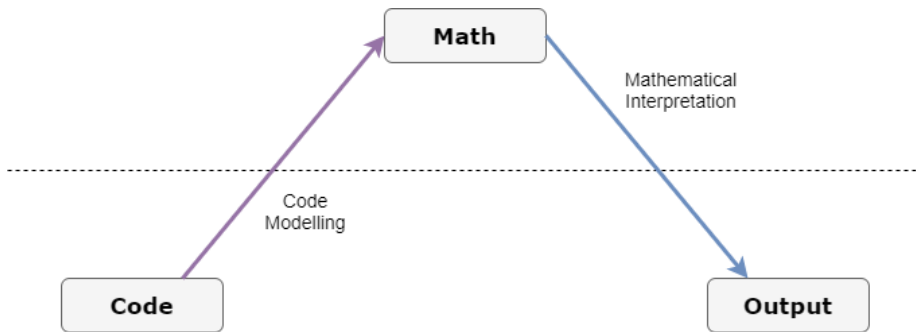


Figure II.20: Replicating Program, as seen in Case A.

In Case B, we saw Rita and Lena use mathematics and computing flexibly. They attempted to use mathematics to find the optimal parameters for their program, and with support were able to find a numerical solution when an analytical one was not feasible. They went through Output Modelling and Math Implementation in that order, and the program closed the circle by transforming

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

Code to Output. This demonstrates the existence of the circular process seen in Figure II.21, which we have named *Improvement Cycle*:

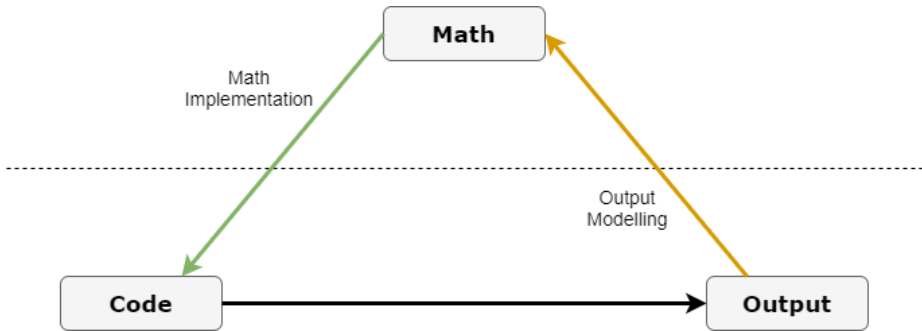


Figure II.21: Improvement Cycle: Code produces Output, the students then use Output Modelling to get to Math, and finally Math Implementation is used to modify the Code further, as seen in Case B.

After completing an iteration of that cycle and being surprised by the result, Rita and Lena went back and forth between Math and Output to improve their previous solution and in the process were able to describe why their first choice was not optimal. This finally led them to choose parameter values that resulted in a log function that performed admirably in the test cases with many fewer terms than their initial attempt. Figure II.22 illustrates this pattern, which we have named Justified Improvement.

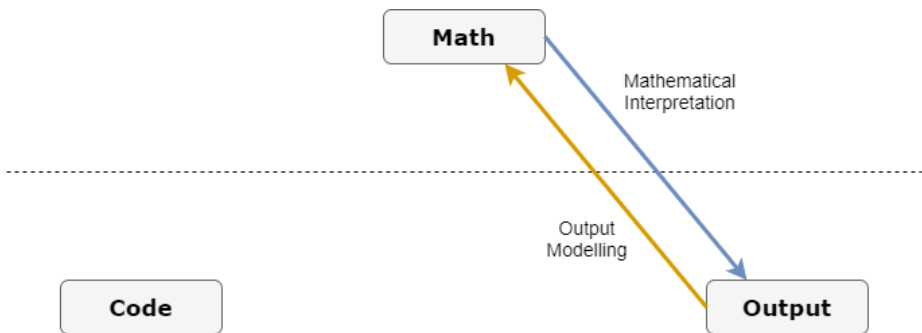


Figure II.22: Justified Improvement, as seen in Case B.

In Case C, we saw Lydia, Martin, and Roger resort to mathematical drawing as a step in the process of implementing the code for numerical integration. This connection enabled them to discuss the mathematical method the program was supposed to represent and revealed a conflict between their work and what they remembered from the lectures about Euler's midpoint method. The resolution came with the realisation that there are two midpoint methods that they had

confused, and the case ended with the students reinforcing connections between what they were doing and their mathematics classes ("this is Riemann").

Lydia, Martin and Roger went back and forth between Code Modelling and Math Implementation. This subset of connections still afforded them to think more deeply about the math they had learned and their organisation of that knowledge. While they finally used the Output for verification, it did not feature in their subsequent mathematical discussions in this case, in contrast to Case B. We have named this pattern Justified Design and illustrate it in Figure II.23.

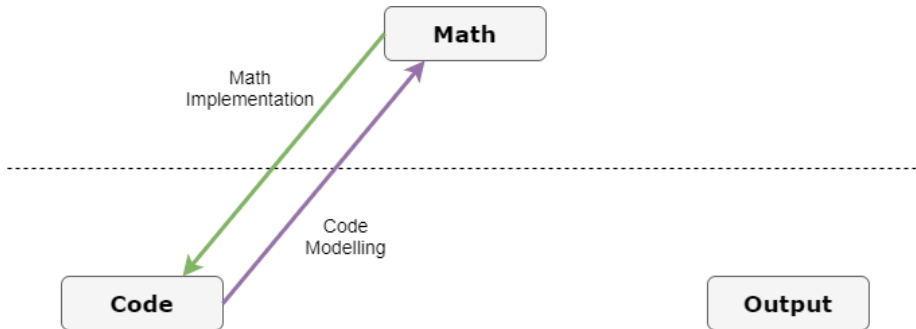


Figure II.23: Justified Design, as seen in Case C.

## II.5.2 Synthesis of Patterns Among Cases

To sum up our findings, these cases demonstrate four different ways that students integrate computing and mathematics (Figure II.20 to Figure II.23). Each of these patterns represents a way that students chained connections together. This, along with the properties in Table II.1 and the connection labels in Table II.2, add a third level of granularity to our classification of the data. On the most detailed level (properties), we examined the properties to determine which connection labels to assign. Then, after the connections were labelled, we identified patterns in connections that occurred dependently on one another. We finally summed up the affordances we found in the data for each pattern, as seen in Table II.3.

Note that all the patterns, except for Replicating Program, were inherently cyclic in our data. Because they ended up where they began, students would be able to repeat these patterns for several iterations if necessary, without additional changes of activity. It is possible that an extra connection from Output to Code could make Replicating Program cyclic in the same way. We do not have sufficient evidence to support this possibility at present, however.

What these cases show us is that working with mathematics and computing in an integrated way has the potential to support students' mathematical reasoning and organisation of knowledge in powerful and flexible ways. These affordances can be seen as a direct result of making cross-domain connections. For a certain class of problems, we have demonstrated that it is possible to design for connections between these domains.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

Pattern	Definition	Case	Main affordances
Replicating Program	Code Modelling → Mathematical Interpretation	A	Understanding a program Formal proof
Improvement Cycle	Output Modelling → Math Implementation (→ running the program)	B	Output closer to standards Explaining process
Justified Improvement	Mathematical Interpretation and Output Modelling (in any order)	B	Output closer to standards Explaining results
Justified Design	Math Implementation and Code Modelling (in any order)	C	Confidence in correctness Organising mathematical knowledge

Table II.3: Connection patterns.

### II.5.3 Discussion

In this paper, we have categorised students' cross-domain connections according to the classification scheme in Table II.2, all of which are based on connections we saw in our data. These connections demonstrate how students see similarities across domains and find these relevant enough to merit a shift in activity or use resources from different domains simultaneously. Our students used mathematics as a resource to write and understand code, and to interpret output. Conversely, they used both code and output as the basis for doing mathematical work. These connections match the descriptions given by (Høffding, 1892) and (Lobato and Siebert, 2002).

Additionally, we recognise the flexible shifts between multiple modes of thinking of (Karakok, 2019) in these connections: Code, Output and Math are the labels we have ascribed to these modes. In Case A we see Gina and Benjamin treat the whiteboard work differently when interpreting it as a representation of the code as opposed to when they treat it as something inherently mathematical. While their shift was perhaps not as flexible as we could have wished for, requiring the interviewer to re-frame the activity for the full connection to be made, the elegant proof they produced suggests that supporting students in becoming more flexible between these modes is something that is worth investigating further. There might well be more modes (subdomains) than we have encountered here, or modes of an even finer grain size, that are useful to us.

What is computing good for in a mathematical context? A clue might be taken from (Greeno et al., 1993), who discussed affordances for student *reasoning* in particular. Even if students arrive at similar answers without using computers,



our results suggest that student reasoning is affected by these shifts between modes of thinking, and not adversely. Our students were able to reason by proving that a program works, explaining both the process of the program's logic and particular results that it produced. Additionally, we saw students organising their mathematical knowledge as a result of reasoning mathematically to write a simple piece of code, suggesting that there is potential value in writing algorithms so that computers may understand them. Of course, it is possible that shifting between modes also adds cognitive load to the students, and for inexperienced programmers especially, this potential may not be immediately accessible when one begins to learn programming in a mathematical context.

The affordances we saw in the interviews appear to be linked to connection patterns rather than individual connections themselves, suggesting that what a single connection affords may depend on the other connections it forms a pattern with. If our analysis is correct on this point, our connection patterns in Table II.3 represent a grain size that is more useful for discussing affordances. We still depend on isolated connections to identify these patterns, however.

One such pattern, the circular process we called the Improvement Cycle is particularly interesting. This pattern suggests that students might even be able to iteratively cycle through a pattern several times as they work toward increased understanding and a better-working program. There is a striking resemblance to the cycle depicted in p. 15 of (Kaufmann and Stenseth, 2020), where "Use mathematics" and "Make hypothesis" could be interpreted as Math, "Change program" and "Test" as Code, and "Observe and analyse" as Output. In Case A and Case C, then, we have seen possible extensions to Kaufmann and Stenseth's model.

## II.5.4 Limitations and future directions

First among the limitations that should be discussed is the possibility that the patterns we observed were due to the tutorial designs and would not occur spontaneously. We acknowledge this possibility, and it remains to be seen how much of what we observed was due to the task, prompting by the interviewer, and spontaneous contributions by the students, respectively. While we claim that the patterns in Section II.5.2 exist and provide affordances for students in some circumstances, we do not claim that they occur spontaneously. More likely, we have taken a first step toward describing contexts in which the patterns surface, which is useful if we find it desirable that students work in these ways.

We also acknowledge that our data set is restricted to a relatively small number of students and contexts, and thus there are limits to what we can claim regarding the generality of our findings. Further research is needed to uncover other possible connection labels and patterns we did not see in our data. We also see a potential for unpacking the causal links between teaching design principles and students' thinking in this context, and we will examine this more closely in a future paper.

It would also be prudent to look more closely at limitations and hindrances to students realising the potential represented by these tutorials. Our data suggests

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

that students who have insufficient prerequisite knowledge may struggle with making these kinds of connections unless they are supported by other students or teachers who can help them apply this knowledge in learning situations.

Our data from Case A suggest that some students, especially when they are new to programming, believe their programs cannot make use of implicit knowledge - all the knowledge available to the program must be there in plain sight. If that were true, it would prevent programs from making use of knowledge that is implicit, such as mathematical knowledge, which may limit the possibilities that these students see. What caused this belief and how to correct it is another direction of future research that may prove useful. More generally, we see potential in investigating students' conceptions of what is possible in mathematics, programming, and the integration of the two.

We also note from our data several instances in which the separation between Math and Code is not so clear. For instance, we had Gina's whiteboard model from Case A, before the students made the full transition to thinking of it as Math, or Lena's initial attempt at solving the equation with Python from Case B, before the students made the connection to loops. We suspect that these "halfway points" will be important stepping-stones for many students in making cross-domain connections and we see potential in investigating these further, both from an education research and a teaching point of view.

In conclusion, we posit that an approach to teaching that integrates mathematics and computing has the potential for making powerful mathematical ideas tangible for learners in ways that do not diminish their richness or relevance. With this study, while we have contributed several examples of such by first-year university students, we cannot claim that our findings are exhaustive. We expect many more such examples to emerge through research and teaching practice in the coming decade.

**Acknowledgements.** This study was funded by the Norwegian Agency for International Cooperation and Quality Enhancement in Higher Education (DIKU), which supports the Centre for Computing in Science Education. We thank Linn Rykkje for very helpful feedback on the manuscript.

## References

- Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, vol. 3no. 2, 115–138.
- Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction*, vol. 16, 68–76.
- Billett, S. (2013). Recasting transfer as a socio-personal process of adaptable learning. *Educational Research Review*, vol. 8, 5–13.

- Broley, L., Caron, F., & Saint-Aubin, Y. (2018). Levels of programming in mathematical research and university mathematics education. *International Journal of Research in Undergraduate Mathematics Education*, vol. 4no. 1, 38–55.
- Brown, J., Stillman, G., & Herbert, S. (2004). Can the notion of affordances be of use in the design of a technology enriched mathematics curriculum?
- Buteau, C., Muller, E., Marshall, N., Sacristán, A. I., & Mgombelo, J. (2016). Undergraduate mathematics students appropriating programming as a tool for modelling, simulation, and visualization: A case study. *Digital Experiences in Mathematics Education*, vol. 2no. 2, 142–166.
- DeJarnette, A. F. (2019). Students' challenges with symbols and diagrams when using a programming environment in mathematics. *Digital Experiences in Mathematics Education*, vol. 5no. 1, 36–58.
- diSessa, A. A. (2017). Conceptual change in a microcosm: Comparative learning analysis of a learning event [Publisher: Karger Publishers]. *Human Development*, vol. 60no. 1, 1–37.
- diSessa, A. A. (2018). Computational literacy and "the big picture" concerning computers in mathematics education. *Mathematical Thinking and Learning: An International Journal*, vol. 20no. 1, 3–31.
- Gibson, J. J. (1979). *The ecological approach to visual perception* [Google-Books-ID: 8BSLBQAAQBAJ]. Psychology Press.
- Gravemeijer, K., Stephan, M., Julie, C., Lin, F.-L., & Ohtani, M. (2017). What mathematics education may prepare students for the society of the future? *International Journal of Science and Mathematics Education*, vol. 15no. 1, 105–123.
- Greeno, J. G., Moore, J. L., & Smith, D. R. (1993). Transfer of situated learning. *Transfer on trial: Intelligence, cognition, and instruction* (pp. 99–167). Ablex Publishing.
- Høffding, H. (1892). *Outlines of psychology*. Macmillan; Company, Limited.
- Inagaki, K., & Hatano, G. (2002). *Young children's naive thinking about the biological world* [Google-Books-ID: cGIBYg7EkEAC]. Psychology Press.
- Karakok, G. (2019). Making connections among representations of eigenvector: What sort of a beast is it? *ZDM*, vol. 51no. 7, 1141–1152.
- Kaufmann, O. T., & Stenseth, B. (2020). Programming in mathematics education. *International Journal of Mathematical Education in Science and Technology*, vol. 0no. 0, 1–20.
- Lavy, I. (2006). A case study of different types of arguments emerging from explorations in an interactive computerized environment. *The Journal of Mathematical Behavior*, vol. 25no. 2, 153–169.
- Lobato, J. (2008, January 1). When students don't apply the knowledge you think they have, rethink your assumptions about transfer.
- Lobato, J. (2012). The actor-oriented transfer perspective and its contributions to educational research and practice. *Educational Psychologist*, vol. 47no. 3, 232–247.
- Lobato, J., & Siebert, D. (2002). Quantitative reasoning in a reconceived view of transfer. *The Journal of Mathematical Behavior*, vol. 21no. 1, 87–116.

## II. Three Cases That Demonstrate How Students Connect the Domains of Mathematics and Computing

---

- Lockwood, E., & De Chenne, A. (2020). Enriching students' combinatorial reasoning through the use of loops and conditional statements in python. *International Journal of Research in Undergraduate Mathematics Education*, vol. 6no. 3, 303–346.
- Lockwood, E., & Mørken, K. (2021). A call for research that explores relationships between computing and mathematical thinking and activity in RUME. *International Journal of Research in Undergraduate Mathematics Education*.
- Mørken, K. (n.d.). *MAT-INF1100 – modelling and computations - universitetet i oslo*. Retrieved January 29, 2021, from <https://www.uio.no/studier/emner/matnat/math/MAT-INF1100/index-eng.html>
- Mørken, K. (2017, September). Numerical algorithms and digital representation.
- Nowell, L. S., Norris, J. M., White, D. E., & Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria [Publisher: SAGE Publications Inc]. *International Journal of Qualitative Methods*, vol. 16no. 1, 1609406917733847.
- Papert, S. A. (1993, August 4). *Mindstorms: Children, computers, and powerful ideas* (2 edition). Basic Books.
- Passey, D. (2017). Computer science (CS) in the compulsory education curriculum: Implications for future research. *Education and Information Technologies*, vol. 22no. 2, 421–443.
- Ramler, I. P., & Chapman, J. L. (2011). Introducing statistical research to undergraduate mathematical statistics students using the guitar hero video game series. *Journal of Statistics Education*, vol. 19no. 3, null.
- Steinberg, R. N., Wittmann, M. C., & Redish, E. F. (1997). Mathematical tutorials in introductory physics [Publisher: American Institute of Physics]. *AIP Conference Proceedings*, vol. 399no. 1, 1075–1092.
- van Someren, M. W., Barnard, Y., & Sandberg, J. (1994). *The think aloud method: A practical guide to modelling cognitive processes*. Academic Press, Inc.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, vol. 25no. 1, 127–147.
- Wolfram alpha. (n.d.).

### Authors' addresses

**Odd Petter Sand** Centre for Computing in Science Education (CCSE), University of Oslo, Postboks 1048 Blindern, N-0316 Oslo, Norway, [oddps@uio.no](mailto:oddps@uio.no)

**Elise Lockwood** Department of Mathematics, Oregon State University, Corvallis, OR 97331, U.S.A. [elise.lockwood@oregonstate.edu](mailto:elise.lockwood@oregonstate.edu)

**Marcos D. Caballero** Department of Physics and Astronomy, Department of Computational Mathematics, Science and Engineering, and CREATE for

STEM Institute, Michigan State University, East Lansing, MI 48823, U.S.A.  
caball14@msu.edu

**Knut Mørken** Centre for Computing in Science Education (CCSE), University of  
Oslo, Postboks 1048 Blindern, N-0316 Oslo, Norway, knutm@math.uio.no



## Paper III

# Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

**Odd Petter Sand, Elise Lockwood, Marcos D. Caballero, Knut Mørken**

*Submitted for publication. Updated preprint available at: <https://edarxiv.org/hrq78>*

### Abstract

We present here the lessons learned by iteratively designing a tutorial for first-year university students using computer programming to work with mathematical models. Alternating between design and implementation, we used video-taped task interviews and classroom observations to ensure that the design promoted student understanding. The final version of the tutorial we present here has students make their own logarithm function from scratch, using Taylor polynomials. To ensure that the resulting function is accurate and reasonably fast, the students have to understand and apply concepts from both computing and mathematics. We identify four categories of such concepts and identify three design features that students attended to when demonstrating such understandings. Additionally, we describe seven important take-aways from a teaching design point of view that resulted from this iterative design process.

### Contents

III.1	Introduction . . . . .	116
III.2	Theoretical Framework . . . . .	117
III.3	The Big Ideas . . . . .	121
III.4	Methodology . . . . .	122
III.5	Initial Design . . . . .	125
III.6	Results, part I: Initial Implementation . . . . .	129
III.7	Second Design . . . . .	136



### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

III.8	Results, part II: Second Implementation . . . . .	140
III.9	Final Design . . . . .	158
III.10	Discussion, Conclusion and Avenues for Future Research . . . . .	160
	References . . . . .	168

#### III.1 Introduction

Our aim with this paper is to showcase the design and implementation of a tutorial that integrates computing with mathematics to strengthen students' understanding of important concepts in both domains. From this, we will articulate important lessons we learned for instructional design. In particular, we will demonstrate that re-creating how the computer performs familiar tasks (such as calculating logarithms) provides rich opportunities for students to make use of mathematics and computing knowledge in an integrated way, as exemplified by one of the key tasks of this tutorial: applying logarithm rules to the representation of real numbers in the computer's memory.

For most students, asking an electronic device for the logarithm of a number can be characterised as a *black box* operation, a term we have borrowed from computer science education (du Bolay et al., 1981): the number is returned as if by magic, with no reference to the means or processes underlying its calculation, nor any measures of its accuracy. While students depend on the correctness of these calculations, there is often little understanding involved beyond figuring out which buttons to press (Gravemeijer et al., 2017; Watters and Watters, 2006). In our tutorial, we offered our students an opportunity to do the authentic work of programming their own logarithmic function using Python and at the same time learn more about the usefulness of Taylor expansions. Doing so offered students an opportunity to reason about the mechanisms and processes behind the calculation of a logarithm, thus deepening their understanding of an important idea at the intersection of mathematics and computing - namely how real numbers are represented in a computer.

We based our tutorials on the framework of Wiggins and McTighe (Wiggins and McTighe, 2005), which is a three-stage process of *backwards design*: first, one attains clarity of the learning goals, and define the understandings that students should come to. Second, one determines what would be acceptable evidence for this understanding having taken place, and design assessments to uncover that evidence. Finally, one designs the learning activities by which the students will be able to uncover the desired understandings.

We elaborate our research questions after defining some key terms and constructs in the following section. Then, Section III.3 goes into the mathematical concepts at the heart of the tutorial. In Section III.4 we describe our methodology, while Sections III.5, III.7, and III.9 detail the three stages of design. In between these stages, we chronologically present our results from each testing (implementation) phase, in Sections III.6 and III.8, before discussing our results and implications for teaching and future research in Section III.10.



## III.2 Theoretical Framework

First, we review the *Understanding by Design* (UbD) framework that we used in more detail. Then, we present a literature review of how computing has been integrated in mathematics education and situate our work in the literature before presenting our research questions.

### III.2.1 Understanding by Design

In the Understanding by Design framework (Wiggins and McTighe, 2005), an *understanding* is defined as a specific and useful generalisation that points to transferable big ideas and requires uncovering and insight to grasp, as opposed to mere drill. We follow (Lobato, 2012) in defining the concept of *transfer* to mean any generalisation students make, without focusing on normative correctness. (Wiggins and McTighe, 2005) echo this sentiment in their discussion of assessment validity: "we typically pay too much attention to *correctness* [in our assessments], and too little attention to the *degree* of understanding" (p. 183). In other words, we often fail to take into account the degree to which performance and understanding are correlated.

Understanding differs from knowledge, but is also connected to it: "An understanding is a mental construct, an abstraction made by the human mind to *make sense of* many distinct pieces of knowledge" (Wiggins and McTighe, 2005, p. 37). In other words, pieces of knowledge are the dots that are connected by the act of sensemaking to form an understanding.

As noted previously, the term "black box thinking" could be applied to situations where a student uses results without understanding the underlying process. Importantly, we should be careful not to say that the student is *unable* to understand, but rather that they have not engaged with how the result was found as something to be understood. In these cases, it may simply be that the task does not require students to attend to this aspect: all that is asked of them is that they get a correct answer without an explanation of how that answer was derived.

Thus, while black box thinking can be said to represent knowledge in the sense that the students know how to formulate a query of the computer to get an answer, understanding in our context means that they also know how the computer finds the answer, and that they are able to interpret and connect it to other forms of knowledge as well. Black box thinking, then, is not what we would consider to be understanding, but it is nonetheless particularly relevant for contexts that involve computing, and we saw an interesting example of this in one of our interviews (see Section 8).

According to Wiggins and McTighe, there are six kinds of understanding: being able to (a) *explain* general ideas, (b) *interpret* specific instances of such ideas, (c) *apply* the ideas and knowing when and how to use them, (d) gain distance to the subject matter and see it from different *perspectives*, (e) have *empathy* with ideas that seem odd or foreign at first glance, and (f) have *self-*

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

*knowledge* so as to know what one knows, what one does not know and how one's learning is progressing.

In the first phase of backwards design, where learning goals are in focus, it is important to prioritise. From least to most important, the curriculum is divided into knowledge that is (a) worth being familiar with, (b) important to know and do, and (c) the big ideas and enduring understandings that everything else hinges on (Wiggins and McTighe, 2005, p. 71). For the latter especially, a set of *essential questions* may be a useful tool for the teaching designer. These are not answerable in finality with a brief sentence but meant to have students ponder them and in so doing uncover the understandings we desire. In short: understandings make use of facts but are not simple facts themselves.

The second design phase focuses on evidence for understanding and assessment, and here it is crucial to distinguish internalised flexible ideas from borrowed expert opinions delivered on cue. This involves supplementing the traditional quiz or test with academic prompts and performance tasks. *Academic prompts*, of which our tutorials are examples, pose questions or problems that require critical thinking, explanations and defence of the answer and methods. *Performance tasks*, on the other hand, ask students to do authentic work that yield tangible products and performances and give students opportunities to personalise the task.

For us to claim that the students understand, they need to provide reasons and support for their choices, in line with the six facets of understanding. It is important that the students' answers are not dependent on blatant cues.

Finally, the third phase of backwards design places the focus on learning activities, of which direct instruction (teaching) is but an example. The optimal designs provide students with engaging and effective tasks. An *engaging* task is recognised as meaningful and intellectually compelling by the learners and presents them with a mystery or challenge they can go hands-on with. *Effective* tasks are ones that help learners become more competent. The goals of such tasks are clear, the criteria are well known, and the students are given opportunities to self-assess along the way.

According to Wiggins & McTighe, a set of design prompts called **WHERE TO** is suggested for analysing the learning activities. These implore the designer to ask whether one has made it clear to the students Where the unit is headed (and Why), Hook (and Hold) their attention, allow them to Experience doing the subject, to Rethink (and Reflect) along the way, to Evaluate their strategies, to Tailor and personalise the task to their own preferences, and to Organise the activity using a whole-part-whole <sup>1</sup> format.

#### III.2.2 Integrated Design: A Literature Review

Integrating mathematics learning with computing is perhaps best illustrated by a counterexample. In what we might call a *disintegrated* design, the students

---

<sup>1</sup>First, one considers the big picture (the subject of the activity taken as a whole), then one goes into the details, and finally one goes back to the big picture and connect the dots between those detailed pieces of knowledge.

learn to code in a context that is mathematical only by accident, with learning activities such as creating a webpage for flight booking with a database back-end. Presupposing then that students know how to program, instructors introduce them to the relevant math software libraries and focus on the *application* of these presupposed skills.

A potential problematic issue with this approach is related to *transfer* of learning: the context in which learning takes place has an impact on both the learning itself and the potential for transfer to other contexts (Billett, 2013). Hence, learning to code in a non-mathematical context plus learning mathematics does not necessarily imply that students will be able to use code effectively *in* mathematics. However, as computational methods are becoming part of the scientific disciplines to an increasing degree (Weintrop et al., 2016), opportunities now exist to have students do authentic scientific work that involves computing.

There are numerous examples in the mathematics education literature of different computational tools being implemented as part of learning activity designs in university mathematics. (Dimiceli et al., 2010) showcase a design experiment where the symbolic Computer Algebra System (CAS) features of the WolframAlpha app were used as an asset in an introductory calculus course. Compared with other CAS software, they found that it had several advantages, although processing power was a limitation. A similar design experiment described in (Caglayan, 2016) demonstrates ways to use the *GeoGebra* dynamic software to visualise Riemann sums, allowing students to visualise and discover important properties of these sums.

Beyond showcasing that designs incorporate these technological tools, there are also studies that investigate the relationship between task design and students' use of them. (Olsson, 2019) comparatively investigated two designs that used *GeoGebra*, in this case a task involving functions designed for schoolchildren in grade 7 to 9. That study, interviewing students in pairs, found that students who were encouraged to explain their thinking performed better overall than students who were encouraged to follow a set of written instructions.

There are also examples of software being designed specifically for educational purposes. One such example is *Grid Algebra* (Hewitt, 2016), a software designed for learners as young as 9-10 years old to visualise the four basic arithmetic operations as movements on a grid when solving linear equations. The software called *Configure* (Greenstein, 2018) similarly lets younger students visualise and conceptualise topological equivalence.

In addition to using pre-existing software and writing dedicated software for educational purposes, there is a third option: having students write or modify computer programs written in a generic programming language. An integrated approach then demands that these programs are written in a mathematical context. One example of this is (Lockwood and De Chenne, 2020), in which students related combinatorial counting problems of different types to corresponding conditional statements in Python programs. This resulted in a reinforcement of conceptual understanding in an area students traditionally have difficulties with, and this reinforcement was attributed to the computational

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

setting.

Another example is the design of a project (Ramler and Chapman, 2011) where students statistically analyse whether players' missed notes in the *Guitar Hero* video game are randomly distributed by writing code in R. In the process of analysing complex data, students would gain hands-on experience using statistical concepts to test their hypotheses (and generally find that the randomness of missed notes depends on the skill level of the player and the difficulty of the song being played). Unlike Lockwood and De Chenne, Ramler and Chapman focus mostly on their design and less on how the setting may influence the reinforcement of concepts. Nonetheless, their design resembles that of the previous example and belongs in the same category.

The work we present here resembles these last two examples and belongs in the same category. Our work is focused on university students using the Python programming language in a mathematical setting (the context is described more closely in Section III.4.1). In this paper, when we say *computing*, we refer to *machine-based computing*, "the practice of developing and precisely articulating algorithms that may be run on a machine" (Lockwood and Mørken, 2021, p. 2). In practice, that means our students use Python programming to articulate, visualise and solve mathematical problems.

(Buteau et al., 2020) point to a central feature of what this mathematical coding entails: To articulate a mathematical process in a programming language, one translates into the language what one would do by hand. To do this, one must realise that the code can indeed work in a similar manner as one does by hand, which is neither self-evident nor independent of what kind of mathematical work one engages in. Integrating coding in mathematics then entails supporting the students in learning, in the words of Buteau et al., "to transform a programming technology into a rich 'mathematical instrument' enabling him/her to [participate] in programming-based mathematical work" (p. 1029).

#### III.2.3 Research Questions

Having described understanding by design and articulated our attention toward integrated design, we now present our research questions. When using the term "understanding" in these questions, we mean specifically the characterisation of understanding given by Wiggins and McTighe discussed previously.

1. Which features of the tutorial design do students attend to when they demonstrate understanding of mathematical concepts?
2. Which mathematical and mathematical-computational concepts did our students demonstrate understanding of?
3. What did we as designers learn about designing for students' understanding from the iterative process of tutorial design?

Note that we focus on cross-disciplinary understanding that integrates mathematics and computing specifically. We chose this because of the sparse

amount of research on mathematical understanding in a computational setting (see Section III.2.2) and because we expect that students' understanding of programming in computational settings has already been amply addressed in computational science research. It might be very interesting to see how the *mathematical* context affects students' understanding of purely computational concepts, but we consider this an avenue for future research.

### III.3 The Big Ideas

In this section, we elaborate on the mathematical concepts that motivated the tutorial design. What were the big ideas (a term borrowed from Wiggins and McTighe) that we wanted our students to engage with?

On the most fundamental level, our tutorials are centred on the following questions: how does one represent mathematical ideas in a computer program? And can one use math to represent computational ideas as well?

More specifically, a common thread running through all our teaching designs are ideas related to the representation of real numbers in the computer's memory. Few rational numbers can be represented exactly in the memory available to represent a number in the computer, let alone irrational numbers. *Rounding errors* are important to be aware of for anyone using computers to model mathematics of science: even though they are often very small errors, they can accumulate and become very large if one is not careful (Mørken, 2017).

The overarching essential questions we ask our students across all our tutorials are:

- How do you know if an error is a rounding error or a math error?
- How do you balance the need for efficiency (speed) with the need for accuracy?

Of the three tutorials we designed<sup>2</sup>, this paper focuses on the second tutorial, which took place in the middle of the semester. This tutorial was the least successful in its initial version (in the sense that the students had a hard time making sense of the tasks and ended up focusing more on the details than on the big ideas). After a redesign, however, it became the tutorial where the students had the most opportunities to work toward and show their understanding of the material. Therefore, it provides valuable insight into the design process, and of all the tutorials, we learned most from the iterative design process of this second tutorial. The tutorial focuses on the following topical essential questions:

- Computers excel at the four basic arithmetic operations, and by extension, polynomials. How do you represent a function like the logarithm using only polynomials?
- What is the point of Taylor expansions? If we can calculate such a polynomial, we already know the precise function!

---

<sup>2</sup>The other two tutorials are described in Paper II.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

In its final form, the tutorial introduction is centred around a single question directed at the students: How do computers and calculators calculate logarithms? As demonstrated in (Watters and Watters, 2006), logarithms are an example of a mathematical quantity that some students use without understanding of the concepts. Even students familiar with the mathematical concept may not have the faintest idea *how* the calculator or computer calculates these numbers. As we shall see, this is not necessarily difficult for the students to grasp, and we expected that many of them would appreciate finding out what happens "under the hood" after using log functions for several years in school.

For many students, the introduction of the Taylor polynomial appears as a solution to a non-problem: How do you approximate a function that is already known? Thus, while most students can calculate Taylor polynomials correctly and perhaps even convince themselves that these polynomials are reasonable approximations, for many the question remains: What is the point, when we already have access to the function itself?

From an educator's perspective, these two challenges can be tackled in concert. If we had to re-invent the computer from scratch, how would we go about programming the first logarithm function? Elder academics may remember how logarithm tables containing some exact values were once used in lieu of modern implements, but how do you go from those select values to an accurate (enough) logarithm for any given number?

The stated learning goals of the final tutorial are that all the students should understand (a) how the computer calculates logarithms, (b) how the way real numbers are represented in the computer can be helpful in this regard, and (c) how Taylor polynomials may be of use to us even if the original function is known. We elaborate on the development of these learning goals in Section III.7.1.

## III.4 Methodology

### III.4.1 Context of Study

A *tutorial* is defined as an educational approach where "instructors are provided with a classroom-ready tool to target a specific concept, elicit and confront tenacious student misconceptions, create learning opportunities, and provide formative feedback to students" (Council, 2012, p. 129).

We designed three tutorials for the first-semester course MAT-INF1100: "Modelling and Computations" (Mørken, n.d.), which is common to mathematics, physics, and electronics students at the University of Oslo (UiO). This course is taught alongside courses in calculus and programming and is intended to link these courses together. This set of courses is intentionally coordinated; typically a mathematical concept is first covered in calculus, then MAT-INF1100 covers how to implement the concept numerically, and, finally, the students write Python code to do just that in the programming class, as described in (Malthe-Sørenssen et al., 2015).

To the best of our knowledge, this particular approach to integrate computing intentionally across the curriculum is rather unique in undergraduate

mathematics education. There are some other models for such integration (for example, coordinating joint computational projects for mathematics and engineering courses at Chalmers University of Technology, Sweden (Enelund and Larsson, 2006; Enelund et al., 2011), and some courses offer explicitly-designed courses to foster (such as the MICA (Mathematics Integrated with Computers and Applications) courses at Brock University, Canada (Ben-El-Mechaiekh et al., 2007; Buteau and Muller, 2017). Furthermore, sometimes mathematics and computing are integrated in a third context, such as bioscience (Nederbragt, n.d.).

However, the UiO model, in which concepts are reinforced across multiple courses and programming is systematically integrated for first year students, is not common. We point this out to provide some overall context for the study we describe, and to situate our design activity within the broader departmental, programmatic, and university systems in which our study took place.

### III.4.2 Data Collection

The study was designed and conducted in five distinct phases:

- The *initial design* phase (spring 2019)
- The *initial implementation* phase, where we performed pilot research interviews one week and observations in classes where the tutorials were used the next (fall 2019)
- The *second design* phase, where we made changes to the tutorials based on lessons learned (spring 2020)
- The *second implementation* phase, where we tested our improved designs in new research interviews, now concurrent with the classes (fall 2020)
- The *final design* phase, where we made final versions based on data from both implementation semesters (spring 2021)

We recruited students in two ways: (a) asking them to volunteer using an online form during one of the first lectures of the semester, and (b) recruiting groups of students during in-class observations (not possible in the second implementation phase due to COVID-19 requirements).

In the initial implementation phase, we conducted 8 interviews with a total of 13 students, of which 5 participated in two interviews and the rest in one. Each interview covered just one of the three tutorials we designed. Gina and Benjamin in Section 0 had already been interviewed using an earlier tutorial, whereas Martin, Lydia and Roger from the same section would be interviewed again using a later one.

In the second implementation phase, we conducted 7 interviews with 7 students, of which 4 participated in three interviews (all the tutorials) and the rest in one. Rita and Lena from Section III.8 were present in all three interviews, and the interview we present excerpts from here was their second one.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

In both phases, the first author interviewed the students and gave them instructions to work together to solve the tasks as they would in class. We captured video of the students and the whiteboard we made available to them, video of their work on the computer, and audio of their conversation. In class, had teaching assistants available to answer questions; in the interviews, the interviewer took on this role if needed. In this way, we were able to resolve any confusion that occurred as a result of the design. This made it easier to use essential questions (see Section III.2.1) as a means to promote understanding without risking that the students got stuck.

#### III.4.3 Analysis

After the interviews were concluded, we had the audio transcribed and selected the most promising episodes as candidates for translation from Norwegian into English. The first author flagged episodes that matched the evidence for understanding that was identified during the preceding design phase (see Sections Section III.5 and Section III.7). The rest of the research team independently reviewed and validated these suggestions.

The first author translated the episodes that all the authors had flagged and enhanced them by inserting images from the video recordings to provide additional insight into how the students worked with the tutorial. The enhanced transcripts were coded according to which type of evidence they provided. On the basis of this coding, we analysed the enhanced transcripts to link this evidence to design features and principles from (Wiggins and McTighe, 2005).

The flagged episodes were then coded by the first author for the kind of understanding students exhibited, and what they paid attention to at the time or just before. Black box thinking was similarly flagged. The rest of the research team then reviewed this coding, using the enhanced transcripts to provide a more complete picture in cases where there was uncertainty which factors affected the understandings and non-understandings students displayed. The tutorial text was also examined to ensure the understandings were the students' own: for example, when students expressed understandings that mirrored text from the tutorial or hand-out, these episodes were omitted for that reason.

The first author then listed the remaining episodes and selected the ones that most clearly impacted the design process or otherwise provided evidence of the tutorials working especially well or poorly according to the learning goals. The justifications for inclusion and exclusion were once again reviewed by the other authors for validation. Even though this paper is an exploratory case study, we wanted to ensure that we picked representative examples that each added something to the take-aways we formulate at the end of the paper.

#### III.4.4 Design Experiments

We classify this study as *design-based research* (DBR), where information about students' learning experiences informs the next cycle of design and instruction. In DBR, one defines pedagogical outcomes and create learning environments that



address them, with special attention given to supporting human interactions. One modifies the process until the desired outcomes are attained and finally reflects on the process to reveal design principles Reeves et al., 2005.

This is not the only way to create research-based teaching. Another alternative is *developmental research*, in which one first defines a problem, then reviews literature and finally settles on the research design Richey and Klein, 2005. We regard this method as less flexible than DBR in the sense that one needs to know where one is going ahead of time. For more exploratory case studies, like those featuring in this paper, DBR allowed the evolving designs to be influenced by data. Additionally, the focus is on the pedagogical outcomes, not research for its own sake. While we do not see these foci as mutually exclusive<sup>3</sup>, we thought it important to keep the students' learning outcomes in focus, especially given the limited amount of literature in the overlap between computing and mathematics (see Section III.2.2. In short, while the goal was a product besides research (the tutorials), we also wanted to use research to make a good product.

DBR excels at addressing complex, often cross-disciplinary, problems. It also lends itself to authentic inquiry-based tasks, is able to reveal new design principles and works on long time scales (two to five years) with continual refinement. We find all of this to be in excellent agreement with the design principles of Wiggins and McTighe (Section III.2.1) and our research questions (Section III.2.3). To identify design features students attend to and describe the concepts they understood, we needed several cycles of design to produce the desired outcomes. Especially the final research question with its focus on extracting design principles from our experience aligns well with DBR.

In the following, we describe the evolution the tutorial which is the focus of this paper through each of the five phases described in the previous subsection.

## III.5 Initial Design

The initial design consisted of the three phases of backwards design as outlined in Section III.2.1: learning goals, evidence, and learning activities.

### III.5.1 Learning Goals

The learning goals for the initial design sought a compromise between several concerns. Firstly, we considered the potential for including a significant computational component that warranted a hands-on approach. Some of the pre-existing course material tended to lean toward the mathematically abstract side of things, and we wanted to bring in the computational domain to a larger extent than before.

---

<sup>3</sup>In fact, we simultaneously performed research on students' integration of mathematics and computing using the same tutorials in Paper II.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

Secondly, we did not want to stray too far from the pre-existing learning goals of the course (as summarised in Mørken, n.d.) so as not to risk the students experiencing the activity as irrelevant.

Thirdly, we wanted to follow Wiggins and McTighe's definitions of an understanding as something that needs to be *uncovered*, not covered; something that makes use of the facts but also explicitly requires the learner to *make sense* of the content (Wiggins and McTighe, 2005).

Finally, we wanted to focus on a topic that students had difficulty with from the traditional approach, in the experience of faculty teaching the course.

We reviewed the learning goals at course level and discussed the focus of the tutorial with faculty teaching the course. They suggested that one topic that students found particularly difficult was the fact that certain functions would display divergent behaviour in certain regions when adding additional terms to a Taylor series: the more terms one adds, the worse the accuracy gets. We found this counterintuitive behaviour to be an excellent example of a concept that required uncovering.

To understand the reason for this strange behaviour, one can consider the definition of the Taylor polynomial:

$$f(x) \approx T(x, a, n) \stackrel{\text{def}}{=} f(a) + \sum_{i=1}^n \frac{f^{(i)}(a)}{i!} (x-a)^i$$

The main factor that assures convergence is the dominant factorial  $i!$ , which ensures that higher order terms will only be small corrections to the preceding terms, even for values of  $x$  far from the point  $a$ . However, for the logarithms and its derivatives, this factor gets cancelled out by a similar factor in the  $i$ th derivative. For the logarithm itself with  $i > 0$ , we have

$$f^{(i)}(a) = (-1)^{i-1} (i-1)! a^{-i}$$

which, when inserted into the Taylor polynomial, yields

$$T(x, a, n) = \ln a + \sum_{i=1}^n \frac{(-1)^{i-1} (i-1)! a^{-i}}{i!} (x-a)^i = \ln a + \sum_{i=1}^n -\frac{1}{i} \left(1 - \frac{x}{a}\right)^i$$

where the factorial in the denominator has been replaced by a simple factor  $i$ . For the absolute value of the next term to be smaller than that of the previous one, we require that:

$$\left| -\frac{1}{i} \left(1 - \frac{x}{a}\right)^i \right| > \left| -\frac{1}{i+1} \left(1 - \frac{x}{a}\right)^{i+1} \right|$$

For large values of  $i$ ,  $\frac{1}{i} \approx \frac{1}{i+1}$ . In this approximation, the convergence we desire is assured by:

$$\begin{aligned} -1 < \left(1 - \frac{x}{a}\right) < 1 \\ 0 < x < 2a \end{aligned}$$

Thus, for any  $x \geq 2a$ , convergence is not assured, and each successive term added runs the risk of the approximation blowing up, diverging more and more from the original function it is supposed to represent. This has profound consequences for our choice of the point  $a$  used to create the Taylor polynomial: if we did not know better, we would be tempted to pick a number like  $a = 1$ , so we could simply drop the zeroth order term (since we then have  $\ln a = 0$ ). What we have just shown, however, is that if we do so, we may have to abandon all hope of an accurate approximation outside the quite narrow interval  $0 < x < 2$ .

In fact, we have shown that for this class of functions, the choice of  $a$  cannot simply be dictated by convenience but must be chosen so that for a maximal input to the function  $x_{max}$ , we want to Taylor expand around the point  $a = x_{max}/2$ . As seen from the remainder when plotted (see Figure III.6), it is desirable to have  $a$  in the middle of the interval. Then, all that remains is to determine the number of terms  $n$  one needs to ensure the required accuracy. For this, one needs only consider the worst-case values  $x = 0$  and  $x = x_{max}$ , since the error (the remainder) grows smaller the closer to  $a$  one comes.

Our learning goals for the initial design were thus that the students (a) should understand that more terms are not always better, (b) that the choice of  $a$  is not always merely a question of convenience, and (c) that while we can make the mathematical error (remainder) arbitrary small, the presence of rounding errors mean that there is a practical limit to how good the approximation can become on a computer.

### III.5.2 Evidence for Understanding

Having defined the learning goals, we moved on to examine what we would consider credible evidence for students understanding the concepts involved. An important limitation of our context is that we were not re-designing an entire course, but rather just designing tutorials to fit into one. Throughout the semester, in each of the three weeks a tutorial would be run, we had access to one hour of the students' time in class and up to two hours for the students that participated in interviews. Therefore, we were not in a position to design formal assessments; whatever evidence we required needed to be part of the tutorial itself.

To that end, we designed the tutorials as academic prompts (see Section 2.1). These require the students to think critically and not just recall knowledge. The focus is on students providing explanations and defending their choice of methods. Typically, these problems are open-ended, which is not the case here: there exists an optimal choice for the pair of parameters  $a$  and  $n$ .

Even so, by requiring that the students use their hands-on experience with the computer program to justify their eventual choice of parameters, we expected that the tutorial worksheets would be able to provide evidence of four of the six facets of understanding as follows: (a) explain (and prove) why the choice of parameters are important, (b) interpret the plots (identifying rounding errors), (c) apply their knowledge by identifying optimal parameters and writing code that measured the error, and (d) display self-knowledge by comparing their

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

initial intuition with experiences from working with the tutorial and reflecting on the process.

#### III.5.3 Learning Activities

The initial tutorial design took a cue from the Maryland Tutorials in physics (Redish, 2009), by having students use their intuition to sort statements about Taylor polynomials into those that they agreed and those they disagreed with. Some of these would be statements that often, but not always, apply, such as "adding more terms increases the accuracy of a Taylor approximation". This task sought to have students think about what they believed to be true regarding Taylor polynomials, and we repeated the same task at the end of the worksheet for comparison. In this repeated task, students were given the opportunity to comment on statements which they had changed their opinion on, or statements which they interpreted differently after having completed the tutorial.

After the initial task, we asked students to find the Taylor expansion of  $f(x) = x^{-2}$ , providing the first few derivatives to enable them to see the general pattern  $f^{(i)}(a) = (-1)^{i+1}(i+1)!a^{-(2+i)}$  that they could plug into the Taylor formula. Next, we asked them to implement the following mathematical functions as Python functions (here presented with correct solutions):

$$\text{taylorterm}(x, a, i) = -\frac{(i+1)}{a^2} \left(1 - \frac{x}{a}\right)^i$$
$$\text{relerr}(y, y_{\text{exact}}) = \left| \frac{y - y_{\text{exact}}}{y_{\text{exact}}} \right|$$

The first function returns term number  $i$  of the Taylor polynomial<sup>4</sup>, and the latter is the relative error, a common measure of accuracy. These functions would then be used by two programs that we provided the students with. These programs plotted the relative error as a function of  $x$  in two different ways: (a) for several different values of  $a$ , keeping  $n$  constant (Figure III.1), and (b) for several different values of  $n$ , keeping  $a$  constant (Figure III.2).

After producing the plots, the students were allowed to change the constant parameter of each program and asked to explain the effects of both parameters. We asked them to interpret what the plots were telling us, especially concerning the rounding errors in the bottom of the plots, and the regions evident in Figure III.2 where more terms make the approximation worse.

Next, we asked them to analyse their own `taylorterm` formula. They had to identify the dominant factor (for large  $i$ ), determine when this factor would grow without bounds. This factor would be the main source of the divergent behaviour and identifying it would also help students identify the region of stability where the expansion always converges.

---

<sup>4</sup>In this first design, we chose  $f(x) = x^{-2}$  so that unlike with the natural logarithm, there would be no need to treat  $i = 0$  as a special case, and to allow for negative values of  $x$ . In retrospect,  $f(x) = x^{-1}$  would have achieved the same, and it would also have been easier to work with for the students.

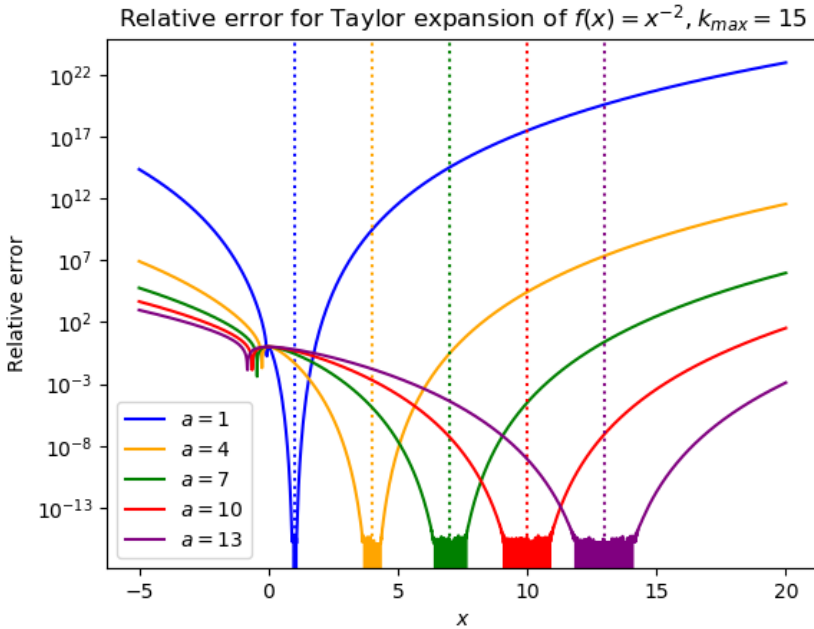


Figure III.1: Relative errors for the Taylor expansion of  $f(x) = x^{-2}$ , with  $n = 10$ , for different values of  $a$ . Originally, we called the number of terms  $k_{max}$ , which confused some of the students, who were used to  $n$  from the lectures. As a result, we changed this in the second design phase (Section III.7). Note that the second axis is logarithmic, hence the "noise" for small values of relative error represents the presence of rounding errors. In this plot, we see indications of higher values of  $a$  resulting in a broader region of high accuracy for this function.

Their next task was to compare the result with the plots and explain how they were connected (the students would be able to see the region of convergence in the plots). Finally, we asked them to explain what it means for a Taylor polynomial to converge. The idea was that they would connect the known concept of convergence for a series of numbers to convergence of Taylor terms in the sum.

As the final step of the design, we used the design standards in (Wiggins and McTighe, 2005) to validate the learning goals, evidence and learning activities.

### III.6 Results, part I: Initial Implementation

In this section, we will describe some results from the first round of implementation. We will present data from two different interviews - one with four students and one with three - and describe what we learned from those interviews. In

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

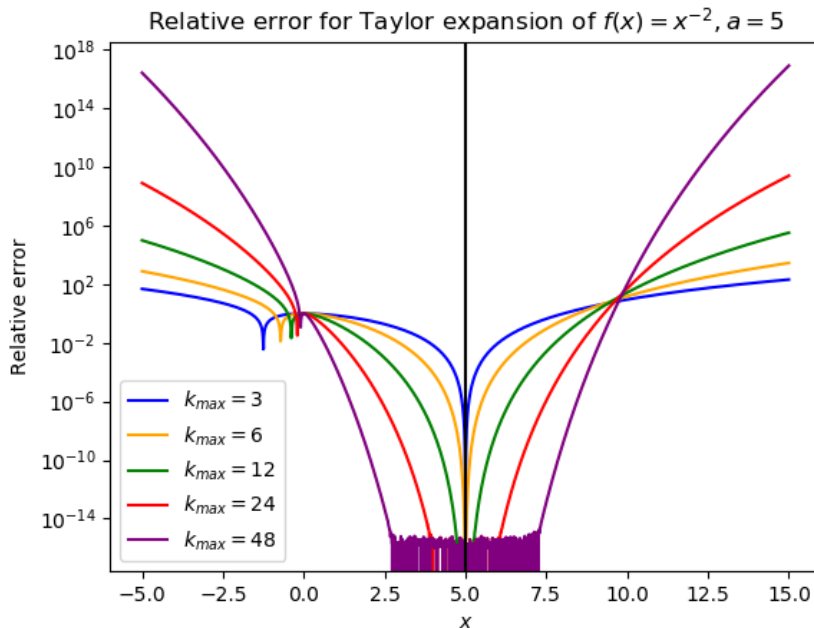


Figure III.2: Relative errors for the Taylor expansion of  $f(x) = x^{-2}$ , with  $a = 5$ , for different numbers of terms. Note the divergence outside the interval  $0 < x < 2a$ , indicated by vertical dashed lines that we added as extra scaffolding for the students.

particular, we will focus on where the tutorial described in Section III.5 fell short of our expectations, and this will form the basis for describing the changes we made based on these experiences in Section III.10.

Once the design was final, we set up interviews with volunteer students recruited from earlier in-class observations of an earlier tutorial, as well as some students that had volunteered to be interviewed at the start of the semester. For this tutorial, we interviewed two groups of students, one with four students and one with three. Due to scheduling conflicts, we had to break up the group of students recruited in class, so that its members were spread over the two interviews. As such, not all the students had worked together before.

The interviews for each tutorial were scheduled the week before that tutorial was to be used in class. This meant the tutorial was tested during the same week the relevant concepts were covered in lectures, as opposed to the week after, which is typically done to give students some time to digest the material before they are given tasks related to it. This is likely to have raised the difficulty for the interview groups, as some students might not have been familiar with Taylor expansions.

The purpose of this was to allow for enough time to adjust the material before subjecting all students to it. The following week, we had planned to let the rest of the class work with the tutorials and perform in-class observations. Unfortunately, the interviews revealed problems with the design that necessitated a re-design from the ground up, as opposed to simple adjustments. As such, this particular tutorial was only tested on the students we interviewed in the first implementation phase, not the class as a whole.

The first group of students consisted of four students: Benjamin, Gina, Martin, and Ruth. Martin had previous experience with computer programming and often suggested what the group should do. Gina was verbally active and asked many questions throughout the interview, often initiating discussion around the mathematical concepts involved. Benjamin and Ruth spoke less than the other two, and for the most part contributed observations and questions without guiding the activity of the group as much as Gina and Martin.

Martin solved the initial mathematical task to find the Taylor expansion almost singlehandedly, as he was the only student who was familiar with the concept. The others expressed that they had not had time to look at the material at the time of the interview. Gina took charge of writing the Python code that reflected Martin's work on the whiteboard. With input from the rest of the group, she wrote the necessary functions and ran the code to produce plots.

As it happened, Martin made a mistake that made it difficult to reproduce the expected results: in the Taylor expansion, he differentiated with respect to the wrong variable,  $f^{(i)}(x)$  instead of  $f^{(i)}(a)$ . We conjecture that this happened because the tutorial used that variable in the example derivatives:  $f'(x) = -2x^{-3}$  instead of  $f'(a) = -2a^{-3}$ . When this happened, Martin was attending to his correct formula for the general Taylor expansion on the whiteboard, and his general formula for the *derivative* (using the inappropriate variable) on his worksheet.

Even though Martin displayed understanding in deriving these general formulas, we interpret that the way he *combined* them on the whiteboard (while saying they could "just plug it in"), is an example of black box thinking. At the same time Martin displayed understanding by being able to explain what he was doing to the other students, which exemplifies that understanding and black box thinking need not be mutually exclusive.

The result was a series of plots that looked almost right, but not quite. In interpreting the tasks, students assumed their plots (such as the one in Figure III.3) were correct, when they did in fact look quite different from the expected results (as shown in Figure III.1):

As a result of this, the interviewer had to intervene and alert the students to the fact that their plots did not look right. A cursory examination of the code did not reveal the error, which was only discovered by the first author after the interview had concluded. In the end, the interviewer had to show the students figures of correct plots that they could use to answer the questions on the tutorial worksheet. As such, the tutorial did not succeed in giving the students opportunities to self-assess, as they had to depend on the interviewer for this.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

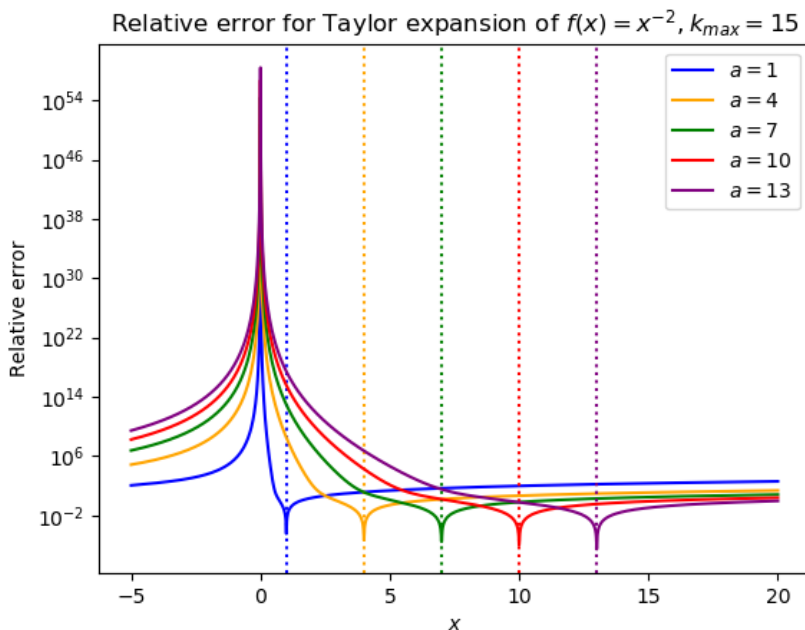


Figure III.3: Plots produced by the students' code in the first phase interview. Note the difference from Figure III.1.

Another problem that required interviewer intervention was the task asking students to analyse which factor in the Taylor terms would dominate when the number of terms grew large. As it turned out, the expression Martin wrote on the whiteboard<sup>5</sup> for Taylor term number  $i$ , which corresponds to

$$T(x, a, i) = (i + 1) (-x)^{-2-i} (x - a)^i$$

did not group all the powers of  $i$  together, but instead kept them as two separate factors. As a result, it was difficult for the students to pick any one of these factors as the dominant one - it would depend on how far  $x$  was from  $a$ . Again, it was demanding for the students to self-assess their expression. They were able to interpret what was asked of them and explain their thinking, but the tutorial did not support them in figuring out what to do when they did not know what to do, to put it in the terms of Wiggins and McTighe.

The interviewer again had to intervene, assuring the students that they could not have known that writing the expression in the way they did would make the task difficult to complete. Gina then transformed Martin's expression into a

<sup>5</sup>The error where he had  $f^{(i)}(x)$  instead of  $f^{(i)}(a)$  is still present in Martin's expression, but that in itself would not have prevented them from completing this particular task.



more usable one on the whiteboard. It turned out that even with an expression that was easier to work with, the students had some difficulty seeing that  $c^i$  would eventually grow larger than  $i$  for any  $c > 1$ :

*Martin:* Yeah,  $2^{10}$  is 1024, but 10 is just 10.

*Gina:* 10 is just... so that makes a lot of sense, OK.

*Interviewer:* What about  $1.01^n$ , is that greater than  $n$ ?<sup>6</sup>

*Martin:* No, that's...

*Gina:* No [writes on whiteboard] eh, oh god... 1.01, was it?

*Interviewer:* Yes.

*Gina:* Raised to the power of  $n$ .

*Interviewer:* Will that also be greater than  $n$  when  $n$  goes toward infinity?

*Gina:* No, because wasn't it, that is, hmm...

*Martin:* That would grow very slowly.

*Gina:* Mhm, slowly.

*Benjamin:* But doesn't it become...

*Gina:* If it's 2 it still becomes [inaudible]

*Martin:* No, it will grow more slowly.

We interpret this exchange as the students focusing on the slow initial growth of  $1.01^i$  compared to  $i$  itself. This seems to have prevented them from seeing (or remembering) that exponential growth will always overtake linear growth at some point<sup>7</sup>. Only when the interviewer pointed this out to them did they agree that 1 was the limit for the exponential term dominating the growth, and that  $0.99^i$  would drop off toward zero as  $i$  grew larger.

While the students in this example were able to justify their answer and generalise knowledge about functions, it also demonstrates that this version of the tutorial often got the students caught up in details rather than pointing them toward the big ideas and essential questions.

In terms of the design principles, it was not clear to them where the instructional unit was headed and why, which would have been necessary for us to claim that the tutorial was effective. This was amply illustrated toward the end of the interview, while asking follow-up questions:

<sup>6</sup>In this version of the tutorial, the number of the current term was called  $k$ . This was later changed to  $i$  so as to be more consistent with the rest of the course material. The students seemed to prefer to call this quantity  $n$ , which would normally denote the total number of terms. This choice was not in conflict with the tutorial, however, which instead used  $k_{max}$  to refer to that quantity.

<sup>7</sup>An example of such a comparison when  $i$  grows large would be that  $i! > c^i > i^m$ , where  $c > 1$  is a real number and  $m > 1$  is an integer. In short, factorials dwarf exponentials, which again dwarf integer powers.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

*Interviewer:* Have you thought about why we are calculating approximate functions that we already have exact expressions for? What is the point of that? We do have...

*Gina:* I'm wondering the same thing, I don't really know, haha. I was thinking...

*Benjamin:* But what is it you use Taylor polynomials for, really?

*Interviewer:* Didn't [the professor] tell you?

*Benjamin:* I didn't catch it but I think [inaudible]

*Ruth:* [inaudible] didn't say that much about it either

*Martin:* Ehm, as long as you don't take a limit where  $k_{max}$  goes to infinity then you have a, is it closed form it's called? Like, one that you can calculate simply, though. [...] A sine [inaudible] has a direct one, or like an expression you can just plug everything into.

*Interviewer:* Shit, that reminds me of something, because if there was no log function in Python, how would you be supposed to calculate it?

*Martin:* Yes. [Gina and Ruth voice agreement]

*Interviewer:* Then you could actually, ah, I have no idea how the logarithm is calculated. But you could use Taylor, but then you just have to know about the limitation<sup>8</sup>, hehe.

*Martin:* But I think it's that way the computer does it with the [inaudible] functions, that they make some approximation or other of the type Taylor polynomials or something, and uses that to calculate...

*Interviewer:* Yeah.

*Gina:* Huh.

*Ruth:* Cool.

The body language and tone of the students toward the end of this exchange suggested to us that they would be interested in trying to make their own log function using Taylor polynomials. This would have provided a means to hook and hold their attention that was missing in this first version of the tutorial. As such, it was not as engaging as we would have hoped. However, while it was unfortunate that there were problematic issues that arose, the initial implementation gave us opportunities to adjust and improve the tutorial.

Before we move on to the next design phase, however, we include an excerpt from another interview, which took place later the same week. We had been able to quickly tweak some of the issues that arose for the first group of students; namely, we updated the example with the wrong variable and provided a plot

---

<sup>8</sup>The narrow area of convergence:  $0 < x < 2a$ .

students could use to check their results against. Note that these were only minor edits and did not constitute a new design phase.

In this interview, three students that we call Roger, Mathias, and Lydia were working on a slightly revised version of the same tutorial. Thanks to these minor revisions, they did avoid differentiating with respect to the wrong variable and obtained the correct plots. But Roger several times expressed a concern about the students having trouble seeing the forest for the trees, as it were:

*Interviewer:* In Exercise 6 we will find out why that happens.

*Lydia:* Yeah

*Roger:* Yes, because I have no idea what we are doing. I don't understand this.

*Lydia:* Because...no, it's a little strange and the values are very confusing, what is  $k$  and what is  $a$  and all that.

A little later, Roger elaborated on this when prompted by the interviewer, connecting his confusion to his lack of understanding of Taylor polynomials as a concept at that point in the semester. Lydia, on the other hand, seemed to have difficulty keeping track of all the variables involved:

*Interviewer:* [to Roger] It's very good that you're saying what you said earlier, and it's helpful if you keep doing that. Eh...

*Roger:* I don't have a complete grasp of what Taylor polynomials are, really.

*Interviewer:* We can have a little de-briefing afterward as well, so that all...

*Lydia:* Mhm [affirmative]

*Interviewer:* ...the threads come together in the end, but...

*Lydia:* Mhm [affirmative]

*Interviewer:* ...part of my point is, if confusion arises, what kind of confusion is it, so it's just good that that is surfacing. We will follow this up afterwards as well, yes.

*Lydia:* It is a little confusing that there are two different plots where one has  $a$  varying and the other  $k_{max}$ . I thought that was...I don't really see it, but yeah.

*Interviewer:* Much information?

*Lydia:* [nodding] Much information.

To summarise, the interview with Gina, Benjamin, Martin and Ruth alerted us to four issues with the tutorial: (a) we misled the students into differentiating with respect to the wrong variable, (b) we assumed the students' Taylor expansion to be of a form that easily would allow them to identify the dominating factor,

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

which we discovered was not always the case, (c) we assumed students are used to reason around identifying with dominating factors, which they may not be, and (d) we lacked a way to hook and hold the students' attention.

Regarding the last issue, we note that Roger and Lydia from the second interview alerted us to a related one: that the tutorial's focus was too narrow and did not point the students toward an issue that was essential from their point of view (What *is* a Taylor polynomial? What is it good for?). In addition, they also expressed that the amount of information the students had to attend to felt overwhelming at times. The tutorial certainly overshot the one-hour target, and as a result of these issues, it was dropped from a wider test run in class the following week. In addition, we flagged this tutorial for a thorough re-design from the ground up.

In spite of these very real problems, the tutorial did succeed in one thing: the students spent a lot of time trying to make sense of what they were doing, and in many instances provided evidence for their understanding by explaining their reasoning. Even though the tutorial was successful in this regard, we still wanted to create a version of it that did not only this, but also gave the students a clear goal, made Taylor expansions seem relevant to them, and did not overwhelm them with information. We also wanted to provide more opportunities to self-assess, so that the students could have realised earlier and by themselves that something was off with their plots.

## III.7 Second Design

During and after the initial round of interviews, we discussed the emerging issues with course instructors and education research colleagues. We compiled the following list of improvements that could be made to the tutorial for the second design phase: (a) to forego the narrow focus on convergence of Taylor polynomials for the bigger issue demonstrating a use for approximating known functions, (b) develop the tutorial to *show* students why Taylor polynomials are useful, not just tell them, (c) provide sufficient scaffolding so that students are not using all their cognitive resources keeping track of what the different variables are, and (d) as far as possible retain students' engagement with the concepts and keep them from using the computer as a black box.

### III.7.1 Learning Goals

An issue with Taylor polynomials that we identified in the first round of interviews (Section III.6) is that students may not understand the point of them: if you already have the exact function, why bother with an approximation that is less accurate? Finding good motivations for Taylor expansions can also be a challenge for teachers (Johnson, 2011). Some even suggest that we might not want to motivate Taylor expansions at all (Šikić, 1990).

One often overlooked point is that in computers, approximations are central to computing most quantities that go beyond the four basic arithmetic operations.

While these operations, that computers excel at, will suffice for calculating any polynomial, other common functions such as exponential, trigonometric and logarithmic functions are not at all trivial for computers to deal with (Arlin, 2012). As such, we depend on these approximations every day, often without being aware of them.

From an educator's perspective, we noticed that the two issues mentioned above are intricately connected. If we imagine, as we did in the interview quoted in the previous section, that we are re-inventing the computer from scratch, how would we go about programming the first logarithm function? The students in the interview responded with interest when confronted with this mystery. What *is* the secret behind this magician's trick?

Taylor polynomials would seem to provide a solution, as polynomials only require the computer to master the four basic mathematical operations. But this simplicity comes at a heavy cost: one must pick a point  $a$  close to the input value  $x$  to create the polynomial, otherwise the number of terms required for sufficient accuracy are prohibitive, even for a modern computer.

To make matters even worse, as we saw in Section 5.1, if the function is a logarithm, we have to pick the point that we expand around with care, or the accuracy may *worsen* as we add more terms. It seems that creating one Taylor expansion for all values of the logarithm would require us to expand around a point so far from 0 that the accuracy would be abysmal for small numbers.

On the other hand, if a way could be found to overcome these difficulties, we would in effect have killed two birds with one stone: Taylor approximations would be useful in a fundamental sense, and the way logarithms are calculated on the computer would become much more transparent to us.

As the second version of the tutorial neared completion, it was suggested by faculty teaching the course that we also include the concept of the remainder in the tutorial. The reasons for this were (a) that students struggled with this concept and could benefit from hands-on experience with it, and (b) that if the students were making their own log functions using Taylor polynomials, the remainder would be useful to them as it provides an upper bound of the error.

We ended up with the following revised learning goals for the second version of the tutorial:

The students should be able to

- explain how computers calculate logarithms
- explain the usefulness of Taylor polynomials for known functions
- explain how the representation of real numbers on the computer is helpful here
- interpret plots of the remainder
- apply what they have learned to pick good parameters
- see the problem from both a mathematical and computational perspective, and be able to merge these perspectives

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

- identify own preference for working computationally or mathematically and self-assess their understanding of either

#### III.7.2 Evidence for Understanding

To ensure that we had evidence for all these understandings, the updated tutorial explicitly asked the students to explain and interpret. A stated goal of the tutorial was that every student in the group should understand what was going on, and they were encouraged to discuss things they were unsure about with the teaching assistants. The final tasks asked the students to reflect on using mathematics and computing together in this fashion.

#### III.7.3 Learning Activities

The new version of the tutorial first introduced the essential questions: how are logarithms calculated, and what are Taylor polynomials good for? Then the students were given functions that calculated the Taylor polynomial and were asked to write a function that calculated the absolute remainder. We listed and explained all the functions and variable involved, to help students not become overwhelmed by all the symbols, like Gina in the previous version.

The starting point for the re-designed tutorial was to figure out how computers actually calculate logarithms of real numbers. The key turns out to be a mapping that reduces the range of the function from all positive real numbers to a very small region. One way to do this is to exploit the way a positive real number is represented as a mantissa and an exponent in the computer (which, incidentally, was already part of the course curriculum),

$$x = M \cdot 2^E$$

where  $0.5 \leq M < 1$  and  $E$  is an integer. We can obtain  $M$  and  $E$  using the Python function called `math.frexp()`. Taking the logarithm of each side and applying logarithmic laws with which the students are familiar, we obtain

$$\ln x = \ln M + E \cdot \ln 2$$

which simplifies things more than is apparent at first glance. If we use Taylor polynomials for our logarithms<sup>9</sup>,  $\ln x$  requires machine-accuracy for all real numbers, whereas  $\ln 2$  is a known constant, and  $\ln M$  only requires us to approximate the logarithm accurately for numbers between 0.5 and 1 (Hammen, 2012).

In other words, this elegant trick not only addressed both the first two bullet points above, but additionally leveraged and made relevant something that was already part of the curriculum: understanding representations of real numbers on the computer. After this was done, we incorporated the concept of the remainder.

---

<sup>9</sup>In practice, approximations requiring fewer terms than Taylor polynomials are used, but the basic idea is the same (Hammen, 2012).

In the course compendium, the absolute value of the remainder can be written in two ways. The first involves an integral, and we judged that adding a second layer of inaccuracy by having the student integrating numerically was needlessly complex. Therefore, we chose the other representation of the remainder of a Taylor polynomial  $n$  terms:

$$R(x, a, n) = \left| \frac{f^{(n+1)}(\xi)}{(n+1)!} (x-a)^{n+1} \right|$$

In fact, the only thing distinguishing this expression from that of the  $(n+1)$  term of the same Taylor polynomial is the replacement of  $f^{(n+1)}(a)$  by  $f^{(n+1)}(\xi)$ , where  $\min(a, x) \leq \xi \leq \max(a, x)$ . We do not know the precise value of this number unless  $x = a$ , in which case the remainder is simply zero (Mørken, 2017, pp. 220-221).

We decided that we could use this limitation to make a point to the students: When in doubt, pick the worst-case scenario. In this case, that meant choosing the  $\xi$  that makes the remainder as large as possible. That way, we do not risk underestimating the error, only overestimating it. In our case  $f(x) = \ln x$ , the absolute remainder becomes:

$$R(x, a, n) = \left| \frac{1}{n+1} \left( \frac{x-a}{\xi} \right)^{n+1} \right|$$

In this case, with  $\xi$  in the denominator, we obtain the largest possible remainder by picking  $\xi = \min(x, a)$ . The resulting expression would be usable by the students to plot the accuracy of their approximation over a range of  $x$  values without resorting to a direct comparison with a pre-existing log function, as in the relative error approach we described in Section III.5. We anticipated that such an approach could support the idea of re-inventing the logarithm from scratch, which our students in the first round of interviews found so intriguing: being able to estimate the error without needing to compare with a pre-existing function would add to the learning activity's authenticity.

To ensure that students got the plots that we expected them to, we gave them a test case using some default parameters to test and self-asses their function with, see Figure III.4. Note that we used  $\ln a$  instead of  $a$  as a parameter for these functions, so that the zeroth term of the Taylor polynomial would simply be the value of that parameter. We can always recover  $a = e^{\ln a}$ , and this approach ensures that we do not have to calculate  $\ln a$  - we can simply *choose* it instead.

When the students had a working remainder function, we asked them to implement their own log function in several steps: (a) provided the representation of real numbers in the computer, find out what happens when you apply the logarithm to such a number, (b) implement the result as a function in Python, and (c) self-assess their work against a test case with the same parameter values as in Figure III.4. We gave the students a machine-accurate value of  $\ln 2$  to use in their function.

Next, the students combined the results of the previous two tasks. They could change the parameters in the remainder plot to find parameters  $\ln a$  and

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

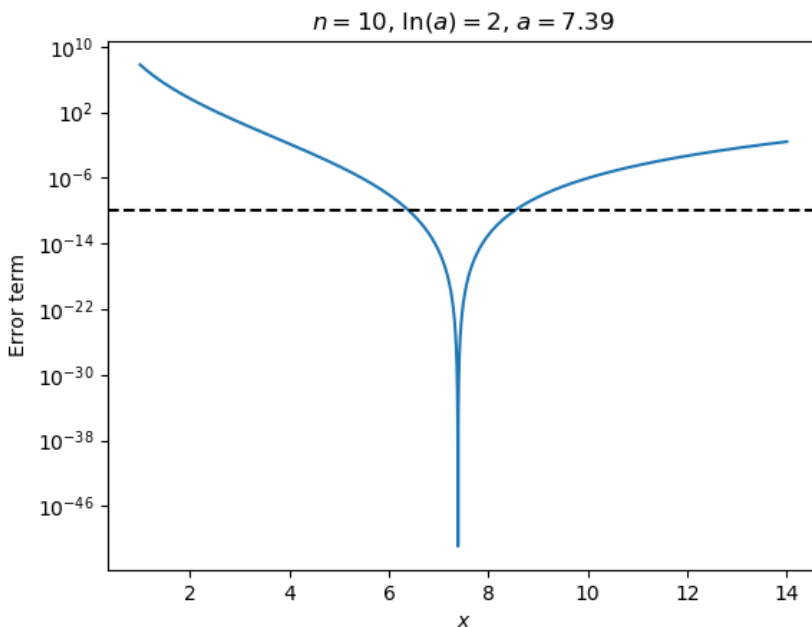


Figure III.4: Test case for the remainder function. If the students' function reproduced this plot, they were free to move on to the next task. Note the slight asymmetry of the curve. The dashed line represents our good-enough threshold value of  $10^{-10}$  for the remainder. In this second version the students had to change the plotted region as well, from (1, 14) to (0.5, 1). In the final version of the tutorial, we instead give them an example in the interesting region (0.5, 1) right off the bat, to give students more time to focus on activities that point toward the learning goals.

$n$  that made the remainder accurate enough in the entire interval between 0.5 and 1. We found that an upper limit of  $10^{-10}$  for the remainder made Taylor expansion accurate enough for our purposes. Then, they were to use these same parameters in their own log function and compare the result with the log function from the standard `numpy` library.

Finally, we asked the students to reflect on the big ideas involved and their own learning.

#### III.8 Results, part II: Second Implementation

We conducted a new round of research interviews with students one year after the initial round. This time, the interviews took place at the same time as the group sessions where the other students worked on the same tutorial in class.



From our experience with the first implementation phase, we deemed the designs to be finished enough that there would be no longer be a need to update them between interviews and classes. This change also helped with scheduling, as we knew the students to be available for interviews at that time and making sure we did not mix students from different cohorts.

We originally planned to interview three groups of three students for each tutorial in the second phase. Due to Covid-19 safeguards, this was reduced to two groups of two students for this particular tutorial. One student was designated to work on the computer, which mirrored its screen on a large external display for the other student and the interviewer to see easily. The other student was designated to work on a large whiteboard when necessary. The interview protocol was largely unchanged from the first phase.

Most of the interesting episodes in the second phase came from our interview with two students whom we will call Lena and Rita. They were both fairly outspoken and took initiative to drive the work forward. Rita expressed a preference for working mathematically over doing programming and chose the whiteboard, while Lena, who worked the computer, had some limited coding experience from high school IT classes that mostly focused on web pages, databases, and the like. Nonetheless, throughout the interview they both contributed substantially to the work on both the whiteboard and the computer.

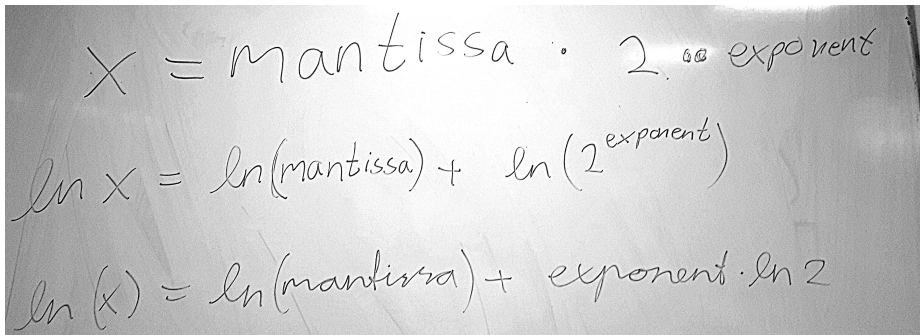
To save time, we provided Rita and Lena with a sheet that contained both the general remainder formula and the formula as applied to the logarithm specifically. Based on this, the students wrote the code in Code Sample III.1, which calculated the value of the remainder and then used provided test code to make a plot that confirmed the correctness of their implementation.

```
def errorterm(x, lna, n):
    a = exp(lna)
    if a <= x:
        xi = a
    else:
        xi = x
    rest = 1/(n+1)*abs(((x-a)/xi)**(n+1))
    return rest
```

Code Sample III.1: Rita and Lena’s implementation of the function that calculated the remainder. They wrote everything except the first two lines, which were given in advance as scaffolding. The function name `errorterm` was in later versions changed to `remainder`, to better evoke the relevant mathematical concept in familiar terms.

Their next task was to use the representation of real numbers on the computer to make their own log function. Rita quickly thought of applying logarithm rules to the exponential representation of a number and wrote down the expression that she had suggested on the whiteboard (Figure III.5). She then asked the interviewer if that was what they were supposed to find. The interviewer confirmed this, and we note that being able to self-assess the results would have been advantageous for the students at this point.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study



The image shows three lines of handwritten mathematical equations on a whiteboard. The first line is  $x = \text{mantissa} \cdot 2^{\text{exponent}}$ . The second line is  $\ln x = \ln(\text{mantissa}) + \ln(2^{\text{exponent}})$ . The third line is  $\ln(x) = \ln(\text{mantissa}) + \text{exponent} \cdot \ln 2$ .

Figure III.5: Rita's usage of logarithm rules on the machine representation of  $x$ . The variable names are the same ones we used in the tutorial text, intended for Python variables. Note that Rita was flexibly able to apply mathematical thinking to these variables nonetheless: she switches from Python syntax to mathematical syntax after the first line.

*Interviewer:* But is it unclear why we wanted to get this expression?

*Rita:* Eh... No. I suppose it is to see something like how much the error increases, perhaps? I think [looks over at Lena].

*Interviewer:* What are you thinking of there?

*Rita:* Well, like, that, I don't...

*Lena:* Like, rounding error?

*Rita:* Well, more like it increases with that fac... [laughs] I don't know for sure. Ehm... Yeah, but sort of if  $x$  is a large number, then in a way... then you go far away from  $a$ , then in a way it becomes a larger number, and then that exponent, then it becomes larger. And then, I'm not really sure, but I'm sort of thinking that then the error increases further.

These last two exchanges reveal that the purpose of manipulating the expression in Figure III.5 was not entirely clear to the students. Even though they were able to do what the task asked them to, they were not sure that they were finished with the task, and we think it likely that they were trying to connect the task to things they had seen in lectures as a result. While Rita's analysis of the error could be the start of a fruitful mathematical argument in itself, it appears that the purpose of this particular task should have been clarified for the students; in other words, they were not entirely sure where the tutorial was headed and why.

Afterwards, Rita and Lena attempted to connect their work on the whiteboard to what they had learned in class and to the plot from their code until the interviewer encouraged them to move on to the next task. They went on to

write the code for the logarithm function, but they ran into an issue with the provided `taylor` function, that calculated values of the Taylor expansion:

*Interviewer:* Is there something that's a little unclear?

*Rita:* We don't understand how we should use the function `taylor` to calculate the logarithm. Does it give... does it return a logarithm?

*Interviewer:* It sh... Yes, that is, it's an approximation... to...

*Lena:* So, if we send  $\ln^{10}$ ...

*Rita:* Oh, yeah! Yes, because the Taylor polynomial is, it is, like, the Taylor polynomial of the logarithm.

*Interviewer:* It would have been nice if that were a little clearer, yes. Good point.

*Rita:* Yes. OK. We... That Taylor polynomial is in a way an approximation of the logarithm function.

*Lena:* OK, yes.

*Rita:* And then, in a way, it is the sum of all the terms in the Taylor function, then.

*Lena:* Yes.

*Rita:* Which is almost the logarithm function, just... except for that remainder that we calculated.

It took only the mention of `taylor` being an approximation to have Rita conclude, correctly, that it was an approximation for the logarithm. Until the interviewer mentioned this, she only referred to this function as the sum of general Taylor terms, not something related to the logarithm in particular. Furthermore, this exchange demonstrates that she was able to explain the relation between this function, the remainder, and the log function the students were tasked to make.

There may be a connection between the tables of variables and functions in the beginning of the tutorial, and this kind of reasoning. Overall, we noted that the students had far less difficulty keeping track of the various variables and functions than the students in the first implementation phase, and we attribute this to these tables and better alignment with the compendium's choices of variable names.

Rita and Lena wrote the function in Code Sample III.2, after some further clarifications from the interviewer. They tested the function and confirmed that it produced the expected result. Next, they proceeded to choose better parameters than the default ones. After some discussion and experimentation with parameters, they found out that  $n = 1000$  made the code run rather slowly. They settled for  $n = 100$  and decided that they wanted the parameter  $a$  in the middle of the interval of interest:

---

<sup>10</sup>Lena's statement refers to the input parameters that gets sent into the function, but this got obfuscated in translation.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

```
def taylorlog(x, lna, n):  
    mantissa, exponent = frexp(x)  
    log2 = 0.6931471805599453  
    lnx = taylor(mantissa, lna, n) + exponent*log2  
    return lnx
```

Code Sample III.2: Rita and Lena's implementation of the logarithm function. They wrote the last two lines, the rest was given in advance.

*Rita:* Maybe it's best to have... that  $a$  is in the middle between 0.5 and 1?

*Lena:* Yeah. 0.75?

*Rita:* Yeah.

*Lena:* And then it's  $\ln 0.75$ ?

*Rita:* I can't do that one in my head. Eh. But this one [*indicates the code*] can calculate that, though.

Here, Rita pointed to a problem with using  $\ln a$  as an input parameter. To get  $a$  in the middle of the interval  $0.5 \leq x < 1$ , the students would need to know the value of  $\ln 0.75$ . They could have found one by trial and error through plotting the remainder (as plots like the one in Figure III.4 do display the value of  $a$ ), but Rita expressed some dissatisfaction with this approach, and stated at the end of the interview that she generally liked precise expressions for quantities.

They tested their choice of parameters and found that they resulted in accuracy well beyond what was needed for the entire interval, as seen in Figure III.6. The interviewer, noticing that the accuracy was in fact well beyond what the tutorial required<sup>11</sup>, challenged the students to find out how few terms they could get away with:

*Interviewer:* How far can we get it to go? With that  $\ln(a)$ ? How few terms can we get away with?

*Rita:* Ehm... I guess one has to... One could calculate it [by hand]. Or, sort of. We could either just try with a lot of different  $n$ 's. [laughs]

*Lena:* And stopwatch and just see how fast it goes? [laughs]

*Rita:* And just take the smallest  $n$ . If not, then you can sort of set that, if you set the remainder as  $10^{-10}$ , then... .

*Lena:* Yeah. Can we manage that?

---

<sup>11</sup>A remainder smaller than  $10^{-30}$ , or 30 decimals would be much more accurate than the 15-16 decimals the computer can accurately represent in 64-bit double precision. Hence, the rounding error that comes from using this precision would be about  $10^{15}$  times larger than the mathematical error in the Taylor expansion itself.

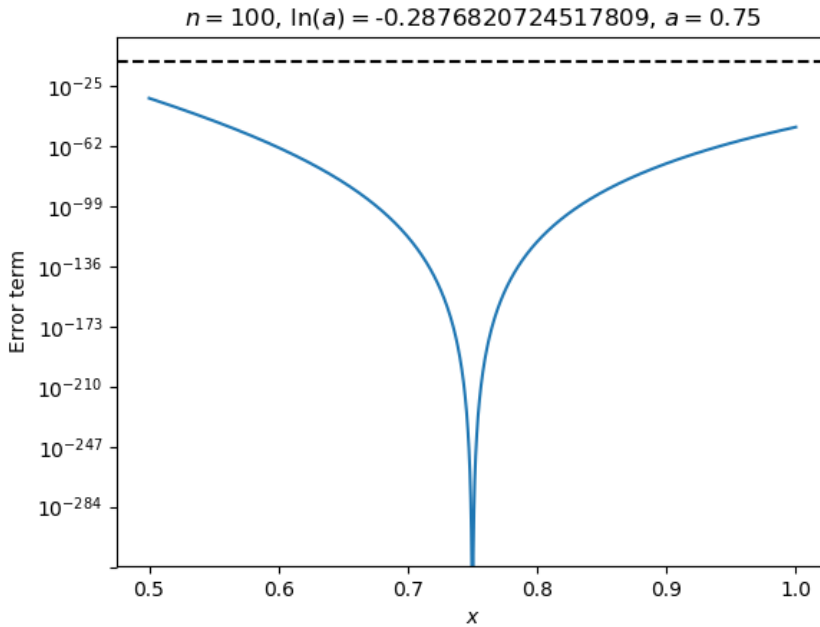


Figure III.6: The remainder plot with Rita and Lena’s initial choice of parameters.

Rita and Lena looked up the mathematical formula for the remainder once more and decided to look for an analytical solution. This was not anticipated: the design had simply assumed that the students would use trial and error and let Python do the heavy lifting (which would indeed be a black box approach). That the students themselves saw a possibility we missed as designers was both a pleasant surprise and a reminder that we should not underestimate our students’ potential appreciation of mathematical rigour.

This impromptu addition to the tutorial (that was later added to the final version we describe in Section III.9) hooked the attention of both students, which certainly fits well with the WHERE TO prompt for designing learning activities in the UbD framework. In terms evidence of understanding, we shall soon see that this challenge got them to apply their mathematical knowledge to the task. Beyond this, there are two features of the design that we claim helped the students in this work and may also have influenced them in believing that such an approach would be fruitful in the first place.

First, we see that the students’ attention was directed to the formula, which provided them with a solution for the remainder in the particular case of the logarithm. Having this formula available, whether it be the result of the students’ own work or, as here, as a provided resource, may have served as a natural entry point. This formula became the right-hand side of the equation they were about

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

to set up.

Second, we also see that they were given a simple, if arbitrary, threshold for the remainder at  $10^{-10}$ , as opposed to not knowing or having to guess. Trusting that this threshold would produce good enough approximations (which we shall see that it did), supported setting up the left-hand side of the equation. Put more concisely, the design offered up quantities that the students were able to assume being equal to each other, and then investigate analytically what followed from that assumption.

Shortly after asking Lena for the remainder formula of the logarithm (provided in the hand-out we gave the students), Rita suggested the following:

*Rita:* Mm. Because if we set it equal to

*Lena:*  $10^{-10}$ ?

*Rita:* Yeah.

*Lena:* Yeah.

*Rita:* And then you calculate<sup>12</sup>? Shouldn't that work? And then you solve it for  $n$ ?

*Lena:* Yeah, and then you get which  $n$  wanted in? Yeah.

*Rita:* [...] Should we try? [...] Or should we just use trial and error?

Rita and Lena seemed a little bit unsure about how to proceed, and the interviewer commented that if they didn't use trial and error, they would need to test for a lot of  $x$  values to ensure that the remainder was below the threshold *everywhere*. Rita then demonstrated her understanding by being able to justify the analytical approach:

*Rita:* Oh yeah, no, but I thought that we just tested for 0.5 and 1 because that's the worst case.

*Interviewer:* You wanted to test the endpoints where, sort of, it's the worst case?

*Rita:* Yeah.

*Interviewer:* That sounds like a really good idea to me.

We interpret this exchange to be afforded by two design features. First, the tutorial explicitly mentioned and explained worst-case thinking in helping students pick a value for the unknown parameter  $\xi$ . This might have influenced Rita's thinking about the current task, as she uses the phrase "worst case" explicitly. Note that Rita is providing evidence for perspective here, as she looks at a problem designed to be solved using computing in a mathematical way.

Second, just prior to asking for the formula, Rita had seen the plot of the remainder for the entire interval and noticed its U-shape (Figure III.6) and

---

<sup>12</sup>The implication that she meant a calculation by hand was obfuscated in translation.

seemed to have interpreted that "higher is worse" in the that plot, allowing her to identify the points that were worse in terms of accuracy than all the rest. In addition to this plot, the students were attending to the example plot in the tutorial (Figure III.4) and had both plots side by side on the computer screen.

Rita attended to the remainder formula and proceeded to set it equal to  $10^{-10}$ . She then attempted to solve the resulting equation on the whiteboard. Unfortunately, the unknown parameter  $n$  ended up in two different places, as seen in Figure III.7, and the students quite correctly concluded that they lacked the mathematical tools to eliminate one of these while preserving the expression's correctness. They tried the common approach of taking the logarithm of both sides to get the unknown quantity out of the exponent, but as a result ended up trapping the other instance of the unknown inside the logarithm:

$$\frac{1}{10^{-10}} = (n+1) \cdot 3^{n+1}$$

$$\ln(10^{10}) = \ln(n+1) + ($$

Figure III.7: The endpoint of Rita's first whiteboard calculation of the number of terms.

The typical way to deal with the latter would be to reverse the process, in effect making it circular without bringing the students closer to a solution. At this point, Lena suggested representing the result of their mathematical work in Python so that they could find a solution:

*Rita:*  $\ln(10^{10})$  equals  $\ln \dots$  But we don't get an expression for  $n$  in this case.

*Lena:* No. [inaudible]

*Interviewer:* What's the problem here? Or what were you about to say, Lena?

*Lena:* Mm. I was about to ask whether it'd be easier to write that formula there [indicates Rita's work] in Python or something. Whether we're able to calculate it in that case.

*Interviewer:* You try that.

This exchange demonstrates that the students were able to justify why they would want to translate a computational problem into an equation and back.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

This fits well with the perspective subcategory of understanding in the UbD framework. We attribute this particular occurrence to the overall framing of the tutorial as a computational activity. We emphasised the computational nature of the work in the design and provided mathematical resources separately from the main tutorial (as a form of appendix). We believe this led the students to expect needing to use Python throughout and have a low threshold for resorting to computational solutions.

In the same way, it is possible that having many more years of experience with mathematics (as opposed to computing) may have equipped them with a low threshold for resorting to analytical mathematics on the whiteboard. However, it should be pointed out that giving the students designated responsibilities - one being in charge of the computer, the other the whiteboard - may have enabled each of them look for opportunities to use "their" tool. If so, it may warrant consideration to make this a permanent feature of the tutorials even after COVID-19 safeguards are no longer required.

Rita and Lena's first attempt at solving the equation using computing was an example of the black-box mentality that we hoped the tutorial would help the students grow out of: Rita asked twice if she could use GeoGebra to solve this equation, claiming that would be the simplest way to find the answer, as you could just write the expression, and that she was skilled in this kind of solution. What triggered the black box approach seemed to be that the students recognised that they were unable to find an analytical solution, by attending to the impossibility of isolating the unknown quantity:

*Rita:* Yeah. GeoGebra can do it for us, I think.

*Lena:* Yeah. So that we don't have to do it by hand if we're not that skilled with calculating logarithms. [laughs]

*Interviewer:* But is there also a different problem that makes this difficult to solve for...?

*Rita:* [indicating her work on the whiteboard] I kind of feel that when we have  $n$  there and... Because now we get  $n + 1$  there, but then we have the logarithm there, and then we need to raise to the power of  $e$ , but then it becomes  $e$  to the power of  $n + 1$  again. And I don't really see how we are supposed to get an expression [inaudible]

*Lena:* No, to get an end of it, sort of.

*Rita:* Yeah.

GeoGebra ("GeoGebra", n.d.), which featured among the examples in Section 2.2, is a plotting software which has been commonly used in Norwegian high schools throughout the 2010s. The approach that Rita described would entail simply writing down each side of the equation and reading off the solution based on the point of intersection, and it is possible that this was Rita's go-to strategy when faced with something she could not solve by ordinary means.

It is important to note that Rita's explanation is correct: this equation is impossible to solve analytically. Its solution is called the product log function



or Lambert W-function (“Wolfram Alpha”, n.d.), and is simply defined as the inverse function of  $f(W) = We^W$ . This is not an explicit definition of the kind these students are used to, and we suspect they would not find it very helpful.

While recognising that the equation could not be solved analytically and that a numerical solution might still be possible demonstrates understanding as in the six facets defined by Wiggins and McTighe (see Section III.2.1), Rita appeared animated when explaining this, and we interpret this as frustration that her mathematical work appeared to be going in circles. She stated in the follow-up questions that she did “like having an expression for things” and seemed to prefer solutions with mathematical rigour. We interpret her preoccupation with using GeoGebra to solve the equation as the only alternative she saw when the equation could not be solved by ordinary means.

Lena’s comment, in contrast, seemed to indicate that a solution might be found by hand, but that she doubted she had the required skill. One could argue that GeoGebra does not *have* to be used in a black box fashion, as the students could have discussed the merits of plotting the solution and found an intersection point more generally, perhaps using Python to do so since GeoGebra was not available during the interview. This, they did not do, and alongside Rita’s statement that “GeoGebra can do it for us”, Lena provided support for our interpretation of this event as black box thinking in the follow-up questions. After the students had completed the tutorial, she linked using computing in mathematics to convenience:

*Lena:* No, really, the thing is that when there are very complicated expressions, or like when we have something raised to the power of 1000 or something like that.

*Interviewer:* Mhm. [*affirmative*]

*Lena:* Then you do quickly think that it can’t be done by hand. But that it’s quite easy to type. And then you get an answer. So then you save some time.

Their first Python attempt went along similar lines, as they translated the equation into their Python editor and tried to find some function that would solve it for them:

*Rita:* How do you, like, get that to solve for  $n$ ?

*Lena:* Yeah. Are there some functions in Python? “Solve equation” or something like that?

*Rita:* Maybe. Try some thing or other. Just write something like “from math import \*”, it’s guaranteed to be a, sort of, math module.

*Interviewer:* [laughs] I think it’s a bit advanced to do that... to solve it symbolically, that is.

*Lena:* OK.

*Rita:* There isn’t anything called “solve”?

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

The interviewer instead suggested a more low-level approach that (a) leveraged Python features the students had already used in class, and (b) focused on *how* Python found the answer rather than the efficiency of finding it. To avoid blatant cues, he described what he wanted the program to do in hopes that the students would recognise it, rather than using familiar terms:

*Interviewer:* But, I do have an idea for how we can do this thing.

*Lena:* Yeah

*Rita:* Yeah, okay.

*Interviewer:* Uhm. The question is whether I can make you think of that same idea without simply handing it to you.

*Lena:* OK. Do you have a hint?

*Interviewer:* A hint. OK. Eh. We could just try different  $n$  values and see whether it becomes  $10^{-10}$  though.

*Lena:* Yeah.

*Interviewer:* In this one [*indicates the equation*]. But it could also be that there is a way to have Python try... a lot of different  $n$  values for us. So that we don't have to try each and every...

*Rita:* Oh, yes.

*Lena:* For loop?

*Rita:* If we write, like, **while** or **for**, yeah.

*Lena:* Yeah. That's not such a bad idea.

*Rita:* No, but then it was how...

*Lena:* OK.

*Interviewer:* Actually, we can also have that *while* stop exactly when we're happy with the [value of the remainder] as well.

*Rita:* [*looks at equation on whiteboard*] Yeah, that's true. If we just write, like,  $(n + 1) \cdot 3^{n+1}$ ...

As this exchange shows, the students responded to the rather explicit challenge of deciphering which programming concept the interviewer was thinking of - a loop. This demonstrates their understanding of a loop as a way for the computer to repeat something with variation. Here, the students showed that they were able to see the challenge of solving the equation from a computational perspective once the relevant concept had been made relevant (and not in an explicit way).

We note that unlike her statement that she was skilled in GeoGebra, Rita stated that she was initially unsure how to implement a similar statement in Python, even after they had translated the idea of a numerical solution to a loop. When the interviewer built on their suggestion by suggesting a **while** loop in particular, and by pointing their attention to the fact that they could decide

when the loop should stop, Rita was able to convert her whiteboard equation to a conditional statement that decided when the loop would finish.

Even though conditional statements are written so that the loop continues as long as it evaluates to True, it is often useful to think about it inversely as a stopping condition: we want the loop to stop when we are satisfied with the remainder. Hence, we want the expression to evaluate to False at that point, and not before. Thus, the conditional we desire is one that evaluates to True as long as the remainder remains too large. While the interviewer might have to take the credit for highlighting this idea, attending to the interviewer's hints and the equation on the whiteboard led Rita and Lena to demonstrate their ability to interpret the situation and apply their programming knowledge by translating the equation into the inequality that we see in Code Sample III.3.

```
n = 1
while ((n+1)*3**(n+1) < 10**10):
    print(n)
    n += 1
```

Code Sample III.3: Rita and Lena's code to solve the equation computationally.

The students wrote a short program containing a *while* loop that would start at the smallest possible number of terms  $n = 1$  and increase the number of terms<sup>13</sup> until the expression Rita mentioned was small enough, demonstrating that they were able to go from the inverse suggestion to a working Python loop. It is important to note that Lena used a version of Rita's expression from the whiteboard,

$$(n + 1) \cdot 3^{n+1} < 10^{10}$$

to set up the conditional, not the provided expression for the remainder itself. Thus, they chose a representation without fractions, absolute values, and negative decimals, demonstrating an application of mathematical skill in a computational environment, which once again demonstrates that the students can take different perspectives to the problem.

Furthermore, the equation had now been transformed into an inequality that would be satisfied as long as the error was too large and was therefore appropriate to use as the loop's conditional statement. We interpret this transformation as further proof of the students' mathematical and computational understanding of the task they were engaged in. However, there are other possibilities for their writing the inequality in this way, and since we did not ask them specifically about this during the follow-up questions, we cannot know for certain.

Without Rita's prior work on the whiteboard, we expect this to have been more involved. Taking a black box approach and translating the formula directly

<sup>13</sup>The interviewer had not suggested that the loop run through values of  $n$  in ascending order, but it appears that the interviewer's suggestion of the loop stopping when the remainder was good enough was sufficient to have the students start with as few terms as possible (ideal if it satisfies the requirement), and gradually move on to a larger number of terms. At this point it would have been interesting to ask the students what they expected would happen to the remainder as the number of terms increased (as the method does not depend on the remainder decreasing with more terms).

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

to Python might have resulted in a conditional statement like the one in Code Sample III.4.

```
n = 1
while (1/(n ** 10))*abs((x - a)/min(x, a))**(n ** 10) >
    10**-10:
    print(n)
    n += 1
```

Code Sample III.4: Equivalent code to Code Sample III.3, without using mathematical work.

This would not have engaged the students mathematically to the same extent but would have them rely on Python functions to do the job for them to a greater degree.

Had the students abandoned Rita's equation as a failed attempt and started from scratch, they might have missed out on the experiences that sometimes computational problems are simplified considerably by first considering them mathematically. Furthermore, in that case, they might not have realised that that the whiteboard math could offer additional insight into the problem, as will be made apparent when we describe the repetition of this calculation that Rita performed soon afterwards.

Running their code, Rita and Lena found that their loop stopped at  $n = 17$ , and changed their earlier code to plot the remainder to use that result as an input parameter. While the result was good at  $x = 1$ , the remainder was still too large around  $x = 0.5$  (see Figure III.8), and the students seemed disappointed with the result until the interviewer offered some encouragement:

*Interviewer:* I thought that worked well, actually.

*Rita:* But it's not beneath  $10^{-10}$ , though.

*Lena:* [inaudible] Yeah. How do you think that works well?

*Interviewer:* Because I'm looking at the point 1.  $x = 1$ . And there it's sort of a hair's breadth below.

*Lena:* Yeah.

*Rita:* Oh yeah, that's what we calculated yes. Yeah, because then we should do that for both sides, then. [...] And then you just take the largest  $n$  value of the two endpoints.

Rita's comments indicate that she not only was able to interpret the output in the context of her earlier whiteboard calculations, but she was also able to look forward and apply that interpretation to formulate a plan for getting the result they wanted.

Our interpretation of these events is that providing the students with code to plot the remainder with the threshold visible in the plot allowed her to realise where along the  $x$  axis there was a problem and to revise their plan to work around the issue. As Rita correctly observed, the number of terms had to be the

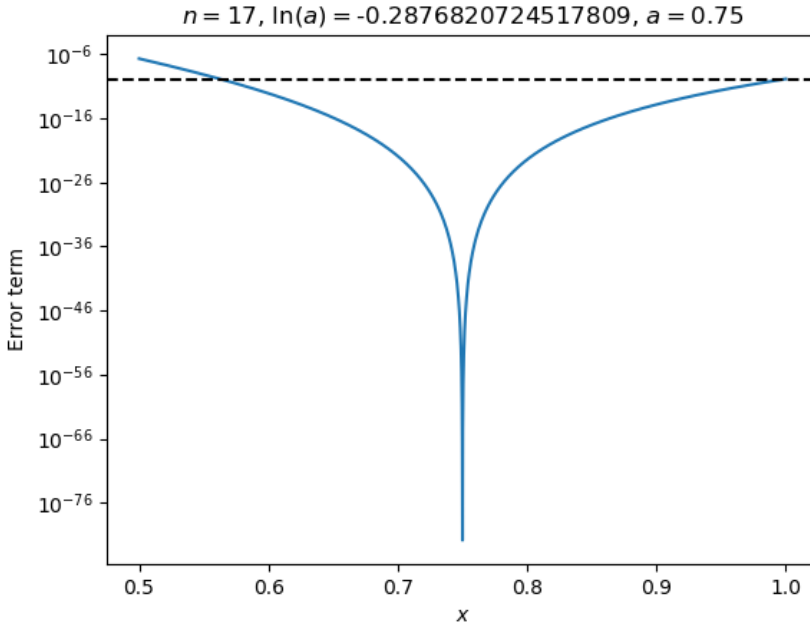


Figure III.8: Rita and Lena’s revised parameter choices, leading to a remainder that was too large near  $x = 0.5$ .

larger among those required by each of the two endpoints. In their earlier plots, the slight asymmetry between the endpoints had been visible, but the students had not paid attention to it until this plot highlighted that one endpoint was on the threshold value of  $10^{-10}$  and the other significantly above it.

Rita then repeated her calculation for  $x = 0.5$ , and in the process she was able to explain why that endpoint had a greater remainder when the interviewer pointed something out to her as she was writing on the whiteboard:

*Interviewer:* We forgot something. We forgot  $\xi$ .

*Rita and Lena:* Yeah.

*Interviewer:* Because *that* one gets a different value [inaudible].

*Rita:* Yeah, that’s it! Because then you’re dividing by a smaller number, and then it does become larger...

What Rita was referring to is that the value of the number  $\xi$  in the remainder formula takes on different values for the two endpoints. Since all we know about  $\xi$  is that it resides somewhere in the interval  $\min(a, x) \leq \xi \leq \max(ax)$ , the

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

worst-case scenario that makes the remainder,

$$\frac{1}{n+1} \left| \left( \frac{x-a}{\xi} \right)^{n+1} \right|$$

as large as possible would be to take  $\xi = \min(a, x)$ . For the two endpoints in question that leads to the scenarios in Table III.1.

Table III.1: Effects of changing  $x$  on  $\xi$  and the remainder.

$x$	$a$	$\xi$	Remainder
1	0.75	0.75	$\frac{1}{n+1} \left  \left( \frac{1}{3} \right)^{n+1} \right $
0.5	0.75	0.5	$\frac{1}{n+1} \left  \left( -\frac{1}{2} \right)^{n+1} \right $

While demonstrating that she was able to interpret and explain the situation in this way, Rita was attending to the plot of the remainder (on screen), her earlier equation for  $x = 1$  (still visible on the whiteboard), and the interviewer's reminder about  $\xi$ . Additionally, she took a mathematical perspective to the plot in question.

Rita continued until she arrived at the expression in the lower right cell of Table III.1. Then Lena contributed a suggestion for how to deal with the absolute value in the presence of a negative number:

*Rita:* What do we do now, because now there's a minus sign and then it's a little more difficult.

*Lena:* But, no, but it's inside an absolute value. Doesn't it become positive anyway, then?

*Rita:* Yeah. That's true.

This demonstrates one additional benefit of implementing Rita's work on the whiteboard. Instead of having to use a Python function such as `abs(...)` to deal with the absolute value, they were able to apply their mathematical knowledge to simplify the task and write more readable code. As before, in their finished expression that went into the while loop,

$$(n+1) \cdot 2^{n+1} < 10^{10}$$

they had removed all fractions and negative exponents from the equation for simplicity's sake. While modifying the program, the students first attended to Rita's whiteboard work (repeated at the interviewer's request for  $x = 0.5$ ) and then their old code for  $x = 1$ :

*Lena:* [switching back to the code editor window, containing the code in Code Sample III.3]OK, so now the expression is...?

*Rita:* [moving from the whiteboard to look at the screen] It's just... where it says 3, you just insert 2, like.

*Lena:* Yes. [inaudible] And otherwise everything's the same?

*Rita:* I think so.

This may look like black box thinking at first glance. However, the program that the students were treating as a black box is code they wrote themselves, from scratch. Like Martin in the first version of the tutorial, we conjecture that Rita and Lena *do* understand how their code works, and even though they do not demonstrate this understanding at this stage, the earlier proof of understanding they supplied makes us confident that they could summon this understanding if required.

If the problem had been formulated as a code problem, it is possible that the students would have accepted more convoluted code, as opposed to simplifying what they could before coding. We therefore postulate that having the students take the analytical path as far as they can before being forced to switch to Python is desirable in light of the learning goal that students should be able to see problems from both a computational and mathematical perspective.

This raises an interesting question, however. Is such work devalued (in the students' eyes) by the existence of a way to get around the problem by using various Python functions and packages instead of mathematics? Rita and Lena's initial desire to use `math.solve()` and find a quick solution seems to point *away* from the understandings we seek. If the students see computing primarily as a means to make laborious mathematics more convenient, then it is possible that they would prefer the black box solution.

Their satisfaction with *how* they found their answers might suggest otherwise, though. Rita was able to explain, with some enthusiasm, why the remainder was different between the two endpoints, which we posit she could not have done had they simply translated the formula verbatim into Python. It is possible, however, that these students were somewhat exceptional in their appreciation of mathematical rigour. Rita stated during the follow-up questions that she felt her work was "more correct" when she did analytical mathematics by hand, which seems to suggest that she would prefer going as far as possible with math alone before using computational tools.

This time when Lena had finished updating the program, the loop stopped at  $n = 27$ , which gave the desired result: a remainder smaller than  $10^{-10}$  for all  $x$  in the interval  $0.5 \leq x < 1$ , with as few terms as possible (see Figure III.9):

*Interviewer:* Wow.

*Rita:* Shit, we're smart.

*Lena:* [laughs] Yeah.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

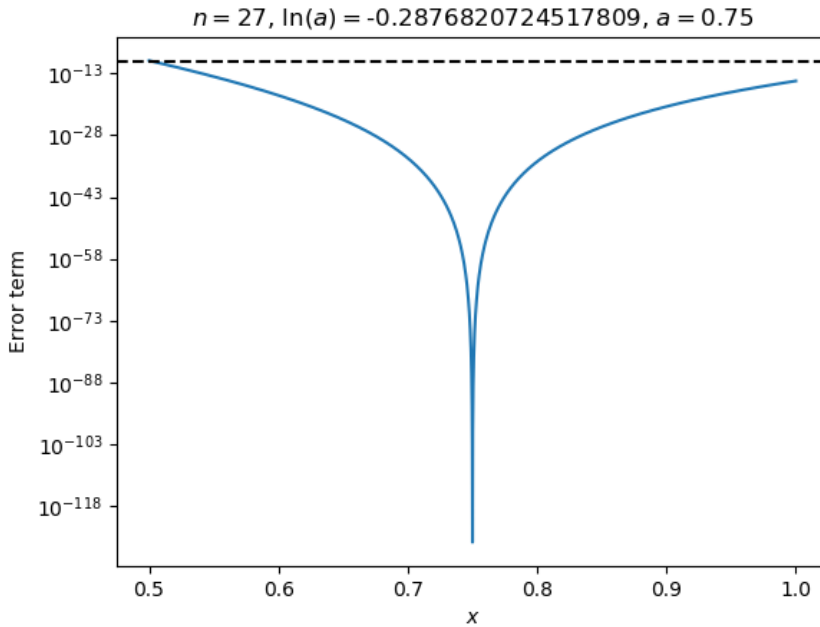


Figure III.9: Rita and Lena's final, optimal choice of parameters.

The interviewer then suggested they try out their log function from the previous task with these parameters, at which point Rita started considering how they could have found the parameter value  $\ln 0.75$  without resorting to *numpy*'s log function:

*Interviewer:* We did choose this one<sup>14</sup> a bit, but...

*Lena:* Well, so it was 27, and 17 for the last one?

*Rita:* But,  $\log 0.75$ , isn't that...no. Kidding. It's the logarithm of  $3/4$ , that. But it isn't any...Because we have...we do have the logarithm of  $1/4$ ...since that is 2 times the logarithm of 2, but we don't have the logarithm of 3.

*Lena:* No.

*Interviewer:* That would be creative too, I didn't even think of that.

Here, Rita demonstrated understanding by explaining how to apply the logarithm rules and what information they would need ( $\ln 3$ ) to find the number

---

<sup>14</sup>The interviewer allowed the calculation of  $\ln 0.75$  using the built-in Python function to save time, after the students had expressed a preference for having  $a$  in the middle of the interval.



they wanted. This reasoning seems to have been prompted by Lena's question and the interviewer's sentiment that this was in a sense cheating, or "cheesing". Her attempt to find this number by other means can be interpreted as a desire to re-create the work of the original log function designers in a self-sufficient way, without having a previous log function available.

To summarise, we found that quite a few features of the new version aligned with the design standards of the UbD framework, which one may use to evaluate teaching designs.

Firstly, Rita and Lena seemed to engage with the challenge of redesigning the original computational log function from scratch. The idea of building it using only the four basic arithmetic operations and mathematical insight appealed to them as a worthy accomplishment, and they suggested many of their own ideas throughout the interview.

Secondly, they were frequently able to demonstrate understanding by explaining concepts (to each other and the interviewer), interpreting plots, code, and mathematics, and apply their knowledge of computing and mathematics.

Thirdly, the list of variables at the beginning and/or these variables' familiar names resulted in little confusion as to what the variable represented. The only difficulty of note was that they sometimes found it hard to separate  $a$  from  $\ln^?(a)$  in the sense that they occasionally mixed them up.

Fourthly, the interviewer's on-the-fly suggestion that the students find the *optimal* value of  $n$ , and the students' subsequent analytical, then computational, work to identify this value hooked the students' attention and gave the students valuable experience with integrating mathematical and computational work. The "worst-case" element of the remainder and the plots seemed to assist the students in this, and the end result was a simpler form of the equation with no fractions or absolute values, which is in line with expert behaviour.

Finally, the students were able to explain why one endpoint of the interval had a higher remainder than the other, which from a UbD perspective is more valuable than a black box solution that provides the right answer without any such explanation.

However, there remained some issues that we flagged to be of interest for the final design phase: (a) the students still spent close to two hours on what we hoped would be a 1-hour tutorial, (b) the purpose of the transformation  $\ln M + E \cdot \ln 2$  was not entirely clear to the students, nor did they know when the expression was finished enough to move on, (c) the role of the computational function `taylor` to provide approximate logarithmic values was not sufficiently clear, (d) using  $\ln a$  as an input parameter made the students want to find the optimal value  $\ln 0.75$ , but the tutorial did not support this in any meaningful way, and (e) some scaffolding, building on the interviewer's rather general cues, seemed necessary to break the habit of students looking for black box solutions to problems that are analytically unsolvable (or inconvenient).

Additionally, one open question still remains to be answered. In particular, we did not ask these students to compare the final form of their equation (the one they implemented in Python) with one that contains fractions and absolute values and relies on Python to do all the work. While the UbD framework

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

encourages opportunities for students to reflect in general, we worry that this might send the wrong message: that the mathematical work was in a sense unnecessary since the black box alternative can be more time efficient. Several of our students appeared to have been primed from high school to look for these kinds of shortcuts that may in fact deprive them of opportunities to understand. As such, we would like to investigate this further in an interview setting before incorporating it into the learning materials.

With these lessons learned in the second round of interviews, we moved to the final design phase.

#### III.9 Final Design

The first major change we made to the final version was to make different versions of it to allow for flexibility with regard to time. We initially designed the tutorial to take students one hour to complete, but in practice, the time was closer to one and a half, in some cases even two. Nonetheless, our data indicated that the time was well spent by the students, therefore we changed the allotted time to two hours.

To also allow for a 1-hour version of the tutorial, which was the original target, we made some tasks optional for the students, at the cost of less opportunity for insight. In Table III.2, we show which tasks are present in each version of the final tutorial, as well as the version that was used in the case that we presented in Section 8. In addition, we suggest some optional exercises that can be used to extend the 2-hour version for further insight if desired. These will be described later in this section.

Table III.2: Versions of the tutorial in Section B. In the 1-hour version, students will have access to all the results they need from the tasks that have been omitted.

Exercise	1-hour version	2-hour version	Interview: Rita and Lena
1	No	Yes	No
2 - 3	No	Yes	Yes
4 - 7	Yes	Yes	Yes
8	No	Yes	No
9	No	Yes	Yes
10	Yes	Yes	Yes
11	No	Optional	No
12	No	Optional	Yes
13	Yes	Yes	Yes

From the interview with Rita and Lena, we learned that the purpose of transforming  $\ln(x)$  using logarithm rules should be better motivated. The students need to know when the task is complete and why we are asking them

to do it. To that end, we added a question to have the students consider the accuracy of `taylor(mantissa)` compared with `taylor(x)`. The students may not be used to reasoning in this way, hence some scaffolding is beneficial to make the purpose of the task clear. This also adds an opportunity for self-assessment, so that they can see whether the result of their work is meaningful.

The next task, asking the students to implement their own log function, also needed revision. Rita and Lena had not realised that the function `taylor` was meant to be used as a stand-in for the logarithm itself. Therefore, we clarified the text of the task to highlight this fact.

The interviewer's suggestion that the students figure out how few terms one could get away with, was met with enthusiasm by Rita and Lena, who used both traditional math and computing to pinpoint this number in a flexible way. We therefore added this as a formal task, encouraging students to use traditional math as far as they are able, then switching to Python if need be. We kept the hint asking if there is a way to have Python test many different values of  $n$ , as that proved effective at getting Rita and Lena to think about loops.

Finding the value of  $\ln 0.75$  turns out to be reasonably simple using the Taylor expansion with  $a = 1$ :

$$\ln 0.75 \approx \ln 1 + \sum_{i=1}^n \frac{(-1)^{i-1} (i-1)! 1^{-i}}{i!} (0.75 - 1)^i = \sum_{i=1}^n -\frac{0.25^i}{i}$$

The students can simply use the `taylor` function with the appropriate parameters to calculate this. We could instruct them to set  $n = 100$ , which by the experience of Rita and Lena is likely to be more than enough to save them the trouble of estimating the optimal number of terms a second time. This demonstrates that Taylor expansions can also be used to find the values of troublesome constants, and that the computer is well suited to work with these expansions<sup>15</sup>.

Considering the length and complexity of the tutorial overall, however, we decided to simply provide the machine-accurate constant for  $\ln 3$  instead. If the students have time to spare, it is better spent working with the optional task of comparing the two remainders at the endpoints and explain why one is smaller than the other. This, as Rita discovered, is due to  $\xi$  taking on a different value when you go from one endpoint to the other. Based on this experience, we believe that asking students to explain this is another way to produce evidence of their understanding.

---

<sup>15</sup>Alternatively, one can write a Python program that looks for a fraction  $-n/d$  that is as close as possible to  $\ln 0.75$  as possible. By starting with  $n = 1$  and  $d = 4$ , incrementing both by one whenever  $-\frac{n}{d} > \ln 0.75$  or incrementing  $d$  by one otherwise, we found that  $-21/73$  is a very good approximation. The downside of this approach is that we needed an accurate log function to estimate the error for each fraction, which is not optimal if the context is the pretence that no such function exists and we are trying to write the first one.

### **III.10 Discussion, Conclusion and Avenues for Future Research**

In this section, we summarise our findings and relate them to the research questions they answer, before presenting conclusions and outlining future possibilities for research.

#### **III.10.1 Summary of findings**

This subsection summarises our findings related to each of our three research questions:

- Which features of the tutorial design do students attend to when they demonstrate understanding of a mathematical concept?
- Which mathematical and computational concepts did our students demonstrate understanding of?
- What did we as designers learn about designing for students' understanding from the iterative process of tutorial design?

##### **III.10.1.1 Features to which students attended**

Our first research question concerned which features of the tutorial design students attend to when they demonstrate evidence for understanding related to mathematical concepts. These understandings, as defined in the UbD framework, were first and foremost of the facets explaining, interpreting, applying, and perspective. The specific concepts they demonstrated understanding of will be covered in the next subsection.

In Section III.6, we saw that when students focused on an example we provided, they explained their thought process clearly; however, the "just plug it in" mindset they displayed is not one we would like the final version of the tutorial to support.

In Section III.8, we found that when the students shifted their attention from how the function `taylor` was defined (sum of Taylor terms) to its purpose (approximating the logarithm), they were able to interpret correctly how to use it and explain how to do so. This shift was prompted by the interviewer merely referring to it as an "approximation".

When the students applied their knowledge of setting up and solving equations, the students attended to the plots  $10^{-10}$  and the remainder formula in concert to set up an equation for the optimal number of terms. In the process, they also demonstrated a remarkable ability to explain and justify their thinking, interpret the possibilities afforded them by these resources.

The same case suggested that black box thinking may occur when the students view a task as impossible, beyond their capabilities, or very tedious to engage with. To overcome this tendency, we had to present them with the possibility of a Python solution that employed computational concepts they could understand.

As they made use of the interviewer's hints, they were able to transform their equation into a program that solved it.

While attending to her own work on the whiteboard, Rita was able to apply the results of this work to justify and explain changes in the program. Finally, we saw Rita explain what additional information they would need to avoid using the pre-defined log function at all, while she focused on the code that Lena had written.

In sum, our students in this tutorial displayed understanding (as defined by Wiggins and McTighe) most clearly when they attended to (a) the *purpose* of code (both their own and the code we provided), (b) their plots (coupled with formulae we provided), and (c) their handwritten mathematical work.

### III.10.1.2 Mathematical and Cross-Disciplinary Concepts

Our second research question deals with the mathematical and mathematical-computational concepts that our students demonstrated understanding of. Here, we found that Rita and Lena in the second round of interviews exemplify the potential of the tutorial to a greater extent than our first-round interviewees. The understandings demonstrated by Rita and Lena broadly fall into four categories: (a) the role of the Taylor expansion and remainder in making a logarithm function, (b) the relationship between mathematics and computing, (c) how to apply worst-case thinking to a continuous interval of numbers, and (d) how logarithmic rules can produce useful transformations.

We note that only the first and partly the second of these overlap directly with the big ideas as we outlined in them in Section III.3. The final two are nonetheless highly relevant additions that we simply did not anticipate in the design stage (Section III.7) but was incorporated into the final design. We will now summarise the concepts in each of these four categories.

In the first category, we saw Rita and Lena explain the purpose of the functions that calculate the Taylor expansion of the logarithm and its remainder. The computer uses the approximation to calculate the logarithm, and we use the remainder to verify that the mathematical error in this approximation is sufficiently low in the entire interval of interest. Therefore, Taylor expansion around a point in the very middle of this interval, as Rita and Lena chose to do, will result in a lower remainder (and potentially fewer Taylor terms) than using other points.

In the second category, Rita and Lena stated a preference for rigorous mathematical solutions and demonstrated that these allowed for greater explanatory power. They recognised the equation as analytically unsolvable, and interpreted the computational concept of a loop to be relevant before translating their equation to an inequality that fit the loop as a conditional statement.

In the third category, Rita and Lena interpreted the plot of the remainder as an opportunity for worst-case thinking: the endpoints of the interval had the largest remainder; hence it would suffice to investigate these. The remainders at the endpoints were not equal, and Rita was able to present an argument for why

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

- namely, because the parameter  $\xi$  changes when moving from one endpoint to another, as seen in Table III.1.

In the fourth and final category, we found that Rita was able to use logarithmic rules to manipulate expressions into more favourable forms. Beyond merely being able to use logarithmic rules, Rita was able to recognise their relevance twice, and in the latter instance explain the motivation for their use.

In addition to the understandings demonstrated by these students, we see potential for more sophisticated understandings of the trade-off between accuracy and efficiency, the general usefulness of reducing intervals when dealing with approximations, and the far from self-evident fact that approximations allow us to represent non-polynomial functions using only the four basic arithmetic operations. As such, the big ideas in Section 3 that we did not see evidence of in the interviews are ideas that we hope the final (and future) versions of the tutorial will target to a greater extent.

#### III.10.1.3 Design Process

Our third research question concerned what we as designers learned from designing the tutorials in this iterative way, with alternating design and implementation phases. In order, we will describe lessons about (a) clarity, (b) student engagement, (c) the knowledge students possessed prior to the interview, and (d) lessons related to mathematics and computing in particular (even though we have also looked at the first three types of lessons in this context).

In the first round of interviews, we discovered that variable names matter. If they are unfamiliar to the students or used in a manner that they are not used to, confusion may arise. These students seemed overwhelmed by all the information. In contrast, the second round of interviews showed very little discussion of what the variables represented, which we attribute to all the variables and functions being presented in orderly tables in the beginning of the tutorial. It is also possible that having the mathematical derivation of the remainder formula in a separate document to be referenced at need helped the students organise the information in a better way.

Even in the second phase, not everything was crystal clear. For instance, the purpose of transforming  $\ln x$  initially proved opaque to Rita and Lena. At first, they had difficulties knowing what they were looking for and why, which resembles the experience of the students from the first phase. We hope that the final version has improved on this with the inclusion of more opportunities for self-assessment and that the purpose of this exercise is now clearer to the students.

As far as student engagement is concerned, the first version of the tutorial lacked a way to hook and hold the students' attention, which would have been in agreement with Wiggins and McTighe's design prompts. Even at the end of the interview, several students expressed that they did not really know what they were doing, or what Taylor polynomials were. In contrast, the quest for reinventing the first log function intrigued students of the first interview.

In the second interview, finding such a function without cheating and using pre-existing log functions held the students' attention to the point that they wanted to adhere more strictly to these "rules" than the tutorial intended, despite a few interventions from the interviewer being required along the way. These students did not wonder out loud what Taylor polynomials were good for, presumably because one explicit goal of the tutorial was to give them an understanding of exactly this. That is not the same as saying they ended up with such an understanding, but it is likely that most of the time, they had a sense where the tutorial was going and why.

The learning goals changed significantly from one version of the tutorial to the next. The first version aimed students toward an important limitation in a concept that they had not yet grasped the full significance of. The second version was redesigned to address two bigger questions that puzzled students in the first round of interviews, and the second round of interviews had students engaging with more central, essential questions than the first: (a) what are Taylor polynomials good for? (b) How was the first logarithm function on a computer made?

When it comes to student knowledge, we note that in order to have the students connect the dots, we must first give them dots to connect. In the first interview phase, the dots proved insufficient. The plots were at times difficult to interpret, and if the students made mistakes, the task got that much harder. In the second phase, providing students with a remainder formula and having them check their code before trusting its plots enabled them to use the information within these to set up an equation that the designers had not anticipated, but ultimately proved very useful to the students.

In the first interview, we made the unfounded assumption that students were able to identify the dominant factor in an expression. It turned out that they were not used to reasoning in this way, and we complicated the matter further by not considering that their equivalent expression ended up looking different from what we anticipated. When the interviewer looked for evidence of their ability to reason in this way, the students only considered the factors which dominated initially, despite the fact that they would later be overtaken by other factors.

Finally, concerning mathematics and computing, we noticed a tendency in the second round of interviews for some students to want their mathematical work to be rigorous, and they appreciated the explanatory power of their whiteboard calculations. It is interesting that the same students were often tempted to use computational tools (both Python and GeoGebra) as a black box. We suspect that these temptations may stem from the way computational tools have been used in Norwegian high school classrooms.

### **III.10.2 Implications for teaching design**

The iterative process of designing and testing this tutorial taught us many things and allows us to formulate seven important take-aways for promoting student understanding in a context that blends mathematics with computing. This is

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

not an exhaustive list, but rather a summary of what we learned from the data provided by our interviews and may as such be sensitive to context.

First of all, we learned about the importance that students know what they are doing and why at all times, as Wiggins and McTighe suggest. In the first version of the tutorial, the students were confused as to what the tutorial wanted them to learn and why it was important in the first place. Our second version improved upon this significantly, and the students working on this version gave evidence of being able to explain, interpret and apply their knowledge on many occasions.

Second, we found that the students' reasoning is often limited by the knowledge they have on hand to reason with. If we overestimate the students' skills and knowledge in mathematical or programming, the result can be that they spend all their time learning or re-learning a concept we took for granted, without being able to use that concept to further their understanding.

Third, we found that black box thinking, where students trust the computer to find a solution without understanding how it is found, occurs when a mathematics task seems impossible, too difficult or very tedious. We suspect, but cannot prove from this data alone, that this behaviour stems from strategies the students have developed during high school mathematics classes. Even for students that prefer and enjoy rigorous analytical solutions, like Rita, black box thinking seemed an acceptable way out of a dead end.

Fourth, we learned that the students use computing in ways that align better with the UbD framework if the problem is formulated in such a way that their computational knowledge seems relevant to the problem. The interviewer's hints, which were far from blatant, got Rita and Lena to restructure their black box problem to one that used trial and error, in a particular order, with a well-defined stopping condition based on their mathematical work. For students with limited computing experience, this kind of scaffolding can lead to greater understanding of both mathematical and computational concepts. In teaching design, we need to support and encourage students to find solutions they understand even when faced with high difficulty problems, as in Rita's case<sup>16</sup>.

Fifth, we note with cautious satisfaction that for the students we interviewed in the second phase, the tutorial hooked and held their attention throughout the interview. It may well be that the students that volunteered are exceptional, or that group composition is key. We might have seen different results if no students in the group felt somewhat confident in doing mathematics or programming. Still, we detected increasing engagement with each version of the tutorial and

---

<sup>16</sup>We noted initially that learning to code in a non-mathematical context while separately learning mathematics does not equate to students being able to use code effectively in mathematics. We claim that Rita and Lena provided many examples of students using mathematics and computation flexibly, as they transformed a computational parameter choice to a mathematical equation, which after some work they again used computing to solve. This process could have ended prematurely with the black box approach, but the way they wrote their program after the interviewer helped them get back on track highlights both the importance of this integrated approach and the fact that we should not expect this to happen automatically in the classroom.



framing the task like a mystery and re-creating the authentic work of early scientists seems to resonate very well with first-year university science students.

Sixth, we learned that students' understanding in this context benefits from opportunities to self-assess, which is also part of the UbD framework. In the first implementation phase, the students had no way of checking their work, and so both the interviewer and the students initially missed the fact that they had differentiated with regard to the wrong variable. In the second phase, the students had plots and screenshots of output to compare their results with.

Seventh and finally, all of the above principles are supported by providing as much clarity in the finished design as possible. We achieved this in our second version by listing and explaining all variables and functions and have since improved upon this further by highlighting the purpose of each function as much as possible. Borrowing vocabulary from computer science ("Implementing an Interface", 1995), we could say that there is a need to separate a function's *interface* (purpose, input parameters, return values) from its *implementation* (how the function works "under the hood"). Using familiar variable names is also helpful to reduce the cognitive load on the students.

To sum this up, the implications for using computing to learn mathematics are that learning activities should be clear and well motivated, with opportunities for self-assessment. It is also important that students' prerequisite knowledge is made relevant, and that the work is made to seem authentic and meaningful as far as possible.

None of the above concerns are unique to a computational setting, however. If students are used to using computers as black boxes that do all the heavy lifting for them, we need to support students in using computing in a transparent way, so that they do not miss out on valuable learning opportunities, like Rita and Lena might otherwise have done. Students might not see how the computational basics they learn in programming courses apply to mathematical or scientific problems, and they should be helped in this both by design and teacher intervention when necessary. A focus on explaining, interpreting and applying knowledge may help prevent black box thinking in our students and allow them to demonstrate creativity like Rita and Lena did with their mathematical approach. We suspect that increased familiarity with programming concepts might help them be equally creative when writing code.

We would like to emphasise that we have not investigated how students perform on these tutorials without teacher intervention. This was a choice we made because students getting stuck and unable to resolve the difficulty on their own would mean less data of lower quality, and as such would probably require at least two more design phases. It remains to be seen to what extent these tutorials can stand on their own. What we have done is try to add the interventions that were helpful to the tutorials themselves to reduce the load on the teacher. Further research and adaptations is required to ensure that students can work on the tutorials without help - at present, we caution against doing so.

### III.10.3 Is Computing Necessary?

We have demonstrated how to design learning activities that support student understanding of mathematical and mathematical-computational concepts in a computer programming context. This begs the question: is computing necessary to achieve these outcomes? Could the students not just as easily demonstrate the same understanding working by hand?

While the question is certainly justified, the point of this paper is not to prove that computing is the *only* way to promote student understanding. It would be interesting to compare and contrast students that use computing to reason with those that do not - it might well be that the reasoning (and the understandings we see evidence of) will be different, even if both groups arrive at the right answers. That is not within the scope of this paper to address, however.

What we *can* say is that we identified features of the computational context that were demonstrably helpful to the students in demonstrating understanding. For one thing, the students attended to the plots they had made as the basis for applying worst-case thinking to a continuous interval of numbers. While we could simply have *given* the students the plots as a resource, that would in essence make them black-box, which we are not sure is desirable. Having students do the authentic work of creating these plots themselves (or part of the work, in this case) is in better agreement with the framework of Wiggins and McTighe.

Furthermore, the computational setting provided a very effective motivation for our students. Its element of mystery and air of authenticity - reproducing the first log function - both depended on the computer as the target of implementing the solution, otherwise the algorithm is simply a mind game, as it would not be practical to employ by hand unless the demands for accuracy and speed were both very low. As far as analysing the accuracy of such an algorithm is concerned, the equation for the optimal number of terms turns out to be insolvable except by trial and error - a process at which the computer excels. A purely by-hand attempt to do the same work would be time consuming and probably frustrating, doubly so if one cannot program a once-and-for-all solution, but has to repeat the work every time a logarithm is needed. In such a case, all semblance of authenticity would be lost.

Additionally, we suspect that the interface-implementation split from Section III.10.2 might be more accessible to students in a computational setting: when writing (or using) a piece of code, it is natural to consider what the code is supposed to *do*, although as we have seen, novice programmers do not always think in these terms. In the literature we found several examples of students and teachers alike wondering what Taylor expansions are good for. This suggests that in mathematical settings, such questions may go unanswered, and computing may be one way (not necessarily the only one) to put these questions front and centre and answer them.

### III.10.4 Future Research

In addition to the findings above, we have identified several promising prospects for future research. The simplest would be to continue with at least one further implementation cycle, if only in the form of classroom observations to see if students using the final version of the tutorial demonstrate additional understandings related to the big ideas outlined in Section III.3. One could also build upon the current work to examine the students' understanding of programming, as we mentioned in Section III.2.3, or compare students using computers with those that do not, as we discussed in Section III.10.3.

One could also investigate further the split between interface and implementation of functions mentioned in Section III.10.2, to find out how common it is that students are unable to recognise what a piece of code is doing because they are too focused on the specific details of how it is done. In computer science education terms, if the students manifest an understanding of all the parts, but not the relations between them, they have difficulties considering a block of code as a whole (Lister et al., 2006). What we would propose, then, is to extend this research to computing in a mathematical context.

Another avenue of future research that we regard as promising is to investigate how widespread black box thinking is among high school students, and how the formation of this mindset is influenced by the way computational aids are used in high school mathematics classes. With the introduction of computer programming into Norwegian school mathematics, students will no longer get a fresh start with computing when they enter university, and as such a comparative study between current and future students might also be possible for a limited time.

To comply with COVID-19 protocols and recommendations, we gave each student in the second phase groups a designated responsibility: one worked on the computer, as Lena did, while the other wrote on the whiteboard, like Rita. It is possible that having each student represent a particular perspective in this way was beneficial to the students, and we cannot rule out that some of the results were affected by this designation. While we also saw examples of Rita using computational language, and Lena mathematical, we see value in investigating the effects of this way of working in a separate follow-up study, either to strengthen the claims in this paper or identify another potentially important design factor.

Finally, we envision studies involving a larger number of students. One way to accomplish this is to put the finished tutorials in a format that allows for assessing evidence of understanding more systematically, building on the UbD framework and the findings of this paper. This would allow for investigation of how widespread these understandings are in the classroom, and how they are distributed among the student population. This would necessitate the design of rubrics to assess to what extends the students understand in different ways, which is something the UbD framework already supports.

**Acknowledgements.** This study was funded by the Norwegian Agency for

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

International Cooperation and Quality Enhancement in Higher Education (DIKU), which supports the Centre for Computing in Science Education. We thank Tor Ole Odden for inspirational input on the tutorial designs and Linn Rykkje for very helpful feedback on the manuscript.

## References

- Arlin, K. (2012). *Calculus - what are the practical applications of the taylor series?* [Mathematics stack exchange]. Retrieved January 13, 2021, from <https://math.stackexchange.com/questions/218421/what-are-the-practical-applications-of-the-taylor-series>
- Ben-El-Mechaiekh, H., Buteau, C., & Ralph, W. (2007). MICA: A novel direction in undergraduate mathematics teaching.
- Billett, S. (2013). Recasting transfer as a socio-personal process of adaptable learning. *Educational Research Review*, vol. 8, 5–13.
- Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. I. (2020). University students turning computer programming into an instrument for 'authentic' mathematical work. *International Journal of Mathematical Education in Science and Technology*, vol. 51no. 7, 1020–1041.
- Buteau, C., & Muller, E. (2017). Assessment in undergraduate programming-based mathematics courses. *Digital Experiences in Mathematics Education*, vol. 3no. 2, 97–114.
- Caglayan, G. (2016). Teaching ideas and activities for classroom: Integrating technology into the pedagogy of integral calculus and the approximation of definite integrals. *International Journal of Mathematical Education in Science and Technology*, vol. 47no. 8, 1261–1279.
- Council, N. R. (2012, May 21). *Discipline-based education research: Understanding and improving learning in undergraduate science and engineering*.
- Dimiceli, V. E., Lang, A. S. I. D., & Locke, L. (2010). Teaching calculus with wolfram|alpha [Publisher: Taylor & Francis, Ltd]. *International Journal of Mathematical Education in Science and Technology*, vol. 41no. 8, 1061–1071.
- du Bolay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices [Publisher: Academic Press]. *International Journal of Man-Machine Studies*, vol. 14no. 3, 237–249.
- Enelund, M., & Larsson, S. (2006). A computational mathematics education for students of mechanical engineering. *World Transactions on Engineering and Technology Education*, vol. 5no. 2, 4.
- Enelund, M., Larsson, S., & Malmqvist, J. (2011). Integration of a computational mathematics education in the mechanical engineering curriculum. *Proceedings of the 7th International CDIO Conference*, 17.
- GeoGebra* [GeoGebra]. (n.d.). Retrieved January 12, 2021, from <https://www.geogebra.org/>

- Gravemeijer, K., Stephan, M., Julie, C., Lin, F.-L., & Ohtani, M. (2017). What mathematics education may prepare students for the society of the future? *International Journal of Science and Mathematics Education*, vol. 15no. 1, 105–123.
- Greenstein, S. (2018). Designing a microworld for topological equivalence. *Digital Experiences in Mathematics Education*, vol. 4no. 1, 1–19.
- Hammen, D. (2012). *Algorithm - how are logarithms programmed?* [Stack overflow]. Retrieved February 1, 2021, from <https://stackoverflow.com/questions/10732034/how-are-logarithms-programmed>
- Hewitt, D. (2016). Designing educational software: The case of grid algebra. *Digital Experiences in Mathematics Education*, vol. 2no. 2, 167–198.
- Implementing an interface.* (1995). Retrieved June 12, 2021, from <https://docs.oracle.com/javase/tutorial/java/landI/usinginterface.html>
- Johnson, J. (2011). *Calculus - the power of taylor series* [Mathematics stack exchange]. Retrieved January 13, 2021, from <https://math.stackexchange.com/questions/73733/the-power-of-taylor-series>
- Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, 118–122.
- Lobato, J. (2012). The actor-oriented transfer perspective and its contributions to educational research and practice. *Educational Psychologist*, vol. 47no. 3, 232–247.
- Lockwood, E., & De Chenne, A. (2020). Enriching students' combinatorial reasoning through the use of loops and conditional statements in python. *International Journal of Research in Undergraduate Mathematics Education*, vol. 6no. 3, 303–346.
- Lockwood, E., & Mørken, K. (2021). A call for research that explores relationships between computing and mathematical thinking and activity in RUME. *International Journal of Research in Undergraduate Mathematics Education*.
- Malthe-Sørensen, A., Hjorth-Jensen, M., Langtangen, H. P., & Mørken, K. (2015). Integrating computation in the teaching of physics. *UNIPED*, 9.
- Mørken, K. (n.d.). *MAT-INF1100 - modelling and computations - universitetet i oslo*. Retrieved January 29, 2021, from <https://www.uio.no/studier/emner/matnat/math/MAT-INF1100/index-eng.html>
- Mørken, K. (2017, September). Numerical algorithms and digital representation.
- Nederbragt, A. J. (n.d.). *BIOS1100 - introduction to computational models for biosciences - universitetet i oslo*. Retrieved February 4, 2021, from <https://www.uio.no/studier/emner/matnat/ibv/BIOS1100/index-eng.html>
- Olsson, J. (2019). Relations between task design and students' utilization of GeoGebra. *Digital Experiences in Mathematics Education*, vol. 5no. 3, 223–251.
- Ramler, I. P., & Chapman, J. L. (2011). Introducing statistical research to undergraduate mathematical statistics students using the guitar hero video game series. *Journal of Statistics Education*, vol. 19no. 3, null.

### III. Students' Development of a Logarithm Function in Python Using Taylor Expansions: A Teaching Design Case Study

---

- Redish, E. F. (2009). *Physics education research group (UMD) / tutorials from the UMD PERG*. Retrieved January 29, 2021, from <http://umdperg.pbworks.com/w/page/10511238/Tutorials%20from%20the%20UMd%20PERG>
- Reeves, T. C., Herrington, J., & Oliver, R. (2005). Design research: A socially responsible approach to instructional technology research in higher education [Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 2 Publisher: Springer US]. *Journal of Computing in Higher Education*, vol. 16no. 2, 96–115.
- Richey, R. C., & Klein, J. D. (2005). Developmental research methods: Creating knowledge from instructional design and development practice. *Journal of Computing in Higher Education*, vol. 16no. 2, 23–38.
- Šikić, Z. (1990). Taylor's theorem. *International Journal of Mathematical Education in Science and Technology*, vol. 21no. 1, 111–115.
- Watters, D. J., & Watters, J. J. (2006). Student understanding of pH: "i don't know what the log actually is, i only know where the button is on my calculator". *Biochemistry and Molecular Biology Education*, vol. 34no. 4, 278–284.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, vol. 25no. 1, 127–147.
- Wiggins, G., & McTighe, J. (2005, January 1). *Understanding by design* (2nd Expanded edition). Assn. for Supervision & Curriculum Development.
- Wolfram alpha. (n.d.).

#### Authors' addresses

**Odd Petter Sand** Centre for Computing in Science Education (CCSE), University of Oslo, Postboks 1048 Blindern, N-0316 Oslo, Norway, [odds@uio.no](mailto:odds@uio.no)

**Elise Lockwood** Department of Mathematics, Oregon State University, Corvallis, OR 97331, U.S.A. [elise.lockwood@oregonstate.edu](mailto:elise.lockwood@oregonstate.edu)

**Marcos D. Caballero** Department of Physics and Astronomy, Department of Computational Mathematics, Science and Engineering, and CREATE for STEM Institute, Michigan State University, East Lansing, MI 48823, U.S.A. [caball14@msu.edu](mailto:caball14@msu.edu)

**Knut Mørken** Centre for Computing in Science Education (CCSE), University of Oslo, Postboks 1048 Blindern, N-0316 Oslo, Norway, [knutm@math.uio.no](mailto:knutm@math.uio.no)







# Chapter 7

## Discussion and Conclusions

### 7.1 Summary of Findings

In Paper I, I showcased an example where computing functioned as a resource for a student in making sense of mathematics and science. I followed up on this work from different perspectives in the subsequent two papers. Before summarising those findings, however, we have time to look back and re-interpret the first paper in light of the theoretical lenses of the last two papers.

In the language of Paper II (summarised in Section 8), the connections in Sophia’s third sensemaking segment could be described as a form of Improvement Cycle that involves both Physics and Math. It would not be a literal cycle as in the two-domain case, because we infer that Sophia ends up in a different domain than the one she started in. I detected no connection from Math to Physics that would allow me to close the circle in this case (see Figure 7.1).

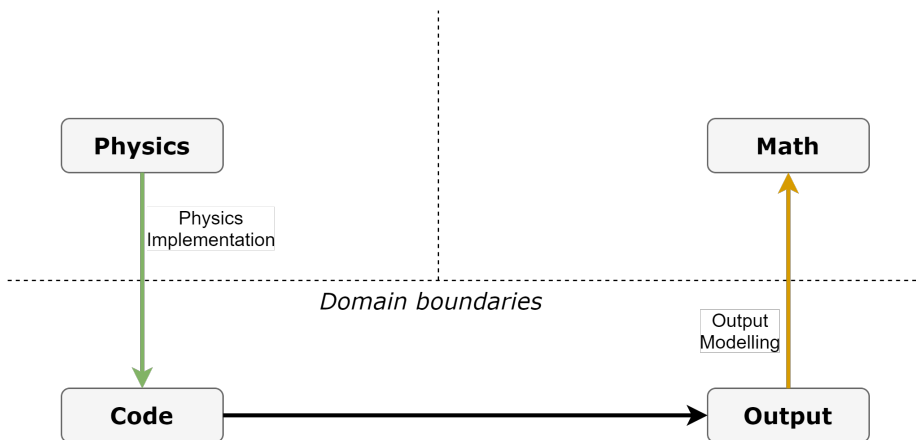


Figure 7.1: Sophia’s third sensemaking segment re-interpreted as a sequence of connections: (1) Cross-domain connection from physics (a single decaying nucleus) to computing (code that would simulate the physical phenomenon), (2) Intra-domain connection between Sophia’s imagined code and the imagined output it would produce, and (3) Cross-domain connection from that imagined output to mathematics (re-interpreting the simplified model as averages).

## 7. Discussion and Conclusions

---

The steps of this pattern are:

- Physics Implementation<sup>1</sup>: she uses computational language to describe how she would implement decay from the perspective of an individual nucleus.
- A connection between code and output that was not named in Paper II, as it would not be cross-domain. In that paper, this connection from code to output was performed by the computer (in Case B). I find it interesting that Sophia was able to (correctly) predict the output of code without writing and running it.
- Output Modelling: she uses the imagined output of that program (that she just described) to re-interpret the simple mathematical model I gave her, now interpreting the numbers as averages.

In the language of Paper III, the learning activity (task) effectively hooked and held Sophia's attention. She almost exited the sensemaking process early, and we interpreted her utterances to mean that she felt this sensemaking was meaningful to her, but perhaps at odds with what the interviewer wanted her to do. She confessed a temptation to "take the easy way out" and just produce an answer, but kept going when the interviewer asked what she would do if she were a researcher and the result mattered to her (see Paper I). This opportunity for personal connection and the element of mystery raised by the apparent contradiction both fit the "Hook and Hold students' attention" criterion in the UbD framework (Wiggins and McTighe, 2005); see also Section 3.1 and (Odden and Russ, 2019).

Similarly, reflecting, rethinking, and revising are also represented in Sophia's work with the task, although the task design does not make the need for these activities explicit. However, I can state more generally that a task designed to allow students opportunities for reflection and rethinking is indeed compatible with sensemaking. If the learning goals require sensemaking for students to come to the desired understandings, this is an element of task design that should not be ignored, and in retrospect, I speculate that Sophia might have been less inclined to take "the easy way out" had I incorporated this design prompt more explicitly.

Furthermore, this "easy way out" can be likened to an example of black box thinking as seen in Paper III. Sophia might have felt that she would not have time to implement the model that she envisioned and thought that finding an answer quickly was the desired outcome of the task. In light of this, her description of the model she would make and imagining of its output represent a compromise that fit within the allotted time set aside for the task.

Finally, I am able to describe the understanding that Sophia demonstrated in the first paper using the six facets of understanding in the UbD framework. Sophia was able to:

---

<sup>1</sup>This would be equivalent to Math Implementation (see Section 8), only drawing on knowledge from a different domain (Physics) instead.

- explain the problems with rounding (hides information) and not rounding (implies fractions of nuclei are possible).
- interpret the mathematical model in light of her imagined computer program.
- apply her knowledge of Python in an appropriate manner to suggest a model whose limit for large numbers coincided with the mathematical one, but also behaved more realistically for small numbers of nuclei.
- regard the problem from a mathematical, physical and computational point of view, and argued for and against these perspectives.

In my Paper II, I identified four patterns in the ways that students connect mathematical knowledge and activities to computational ones. They were able to:

- reproduce or prove a program mathematically to understand how it works,
- iterate in cycles using mathematics to connect outputs to modification in code in order to explain how they are improving the program,
- employ mathematics to decipher program output to justify choices they make, and
- employ mathematics to design a program to organise mathematical knowledge.

In Paper III, I identified six design features students attended to when they demonstrated understanding of mathematics and computing. From the iterative design process exemplified by our Taylor expansion tutorial (Section B), I learned that

- the students need to know where their work is headed, and why,
- the students' reasoning are limited by the knowledge they see as available to reason with,
- black box thinking may occur when a task is seen as impossible, very difficult or very tedious,
- problems need to be framed so that novice programmers see their computation skills as relevant to the problem<sup>2</sup>,
- mystery and authenticity matter to students and can hook and hold their attention,
- students need opportunities to self-assess, and

---

<sup>2</sup>These cues need not be blatant, as we saw with the interviewer's hints to Rita and Lena in Paper III.

## 7. Discussion and Conclusions

---

- all of the above are supported by providing as much clarity and structure for the students as possible.

It is worth noting that the sensemaking framework from Section 3.3 can also be applied to the data from the final two papers, especially Paper II, where the connections students make play an important role:

In the case of Gina and Benjamin (Case A of Paper II), the students realised that they did not understand the program that the first author gave them. Benjamin proposed that the program could be understood as an equation and went about solving it to verify this understanding. As they got  $0 = 0$  as a result, Benjamin used the program output to verify it, and realised there was something more he did not understand when the output did not match the equation they had set up.

In the case of Rita and Lena (Case B of Paper II, also featuring in Paper III), the students realised that the number of terms for the Taylor expansion their script had calculated was not sufficient to get the error low enough in the entire interval of interest. When the interviewer pointed out that the error (the remainder) was just good enough at one endpoint, Rita proposed that they had to use "worst case" thinking and check both endpoints. In doing so, she verified this by re-doing the math, during which she (a) explained *why* the endpoints had different remainders (different values of the parameter  $\xi$ ) and (b) ended up with a plot that showed the remainder was low enough in the entire interval.

In the case of Lydia, Martin, and Roger (Case C of Paper II), the students realised that the midpoint method that they had implemented and tested was different from the midpoint method that they remembered from the lectures. Martin proposed that there was two different midpoint methods, and by discussing and describing the methods, the students were able to remember that Euler's midpoint method is used for differential equations<sup>3</sup>.

In sum, these three papers provide what I hope are valuable example for educators, education researchers, and ultimately students in the context of computing integrated with mathematics and science. To sum up the contribution of each paper in a sentence, Paper I shows how computing may help students make sense of things, Paper II identifies different patterns of connections between mathematics and computing that benefit students' learning, and Paper III identifies design principles that may be applied in an iterative process to further students' understanding.

### 7.2 Limitations

There are some important limitations to my work that I should address before I conclude from the results summarised above. Firstly, with a limited number of students, it is hard to generalise to the entire student population. The students

---

<sup>3</sup>This led to a further sensemaking segment after the interview, in which the interviewer discovered that these methods are indeed related but presented differently in each context. See Case C in Paper II for details.

that volunteered and the interviews we selected for deeper analysis could have been exceptional cases. It may also be that the University of Oslo context is unique enough that these results are not representative of what one could expect for higher education institutions in general.

During the analysis, as the theoretical lenses were chosen and the main foci of the papers became clear, we identified several missed opportunities to ask students follow-up questions. While this is inevitable in exploratory studies such as these, it is important to acknowledge that it is a limitation, albeit one that follow-up studies with predefined foci should be able to remedy.

A third important limitation is that the results may have been influenced by the tutorial designs<sup>4</sup> or interviewer prompting. Therefore, it is important to note that we do not claim that computing caused these results *spontaneously*. More likely, in addition to providing examples of what is possible, we also have some clues as to how one may bring it about.

Finally, the COVID-19 pandemic significantly impacted the amount and quality of data we could collect in the fall semester of 2020. While I have worked hard to make the best of the data that I have, future studies in a less restricted teaching and research environment should be able to collect more data more easily.

### 7.3 The Big Answers

In Section 1.4, I posed the following questions for the project as a whole:

- How do the students themselves integrate science, mathematics, and computing in the context of representing real numbers on the computer?
- What are the resulting affordances for learning?
- How does the design of learning activities support or hinder this integration?

These are big questions, and the answers are not necessarily simple enough to be implemented in a concrete teaching situation without considering the context (more about this in Section 7.4). Nonetheless, I may summarise what I learned about these questions from the three papers.

For the first question I learned that the students make connections between computing and other domains that follow certain patterns. These connections, as we saw in Section 7.1, can make up the steps in a sensemaking process. Examples include students making sense of both mathematical and computational rounding errors, using the representation of real numbers to re-invent the computational logarithm, and finding the sweet spot between accuracy and efficiency that modern science balances on (see for instance Pena-Marín and Yan, 2021).

For the second question I learned that the results of these processes can be understandings that students demonstrate, particularly by explaining concepts,

---

<sup>4</sup>Not really a problem for Paper III since that focused on teaching design, but more important for the first two papers.

## 7. Discussion and Conclusions

---

interpreting complex information, and applying their knowledge from several domains. The results of their work can be as diverse as formal proofs, justified program design, and improved organisation of knowledge. Among the understandings that students demonstrated, the relationship between mathematics and computing appeared in all my papers. In the final paper that probed such understanding on a more detailed level, we saw the students demonstrating understandings of Taylor expansions, remainders, worst-case thinking and logarithmic rules in particular.

For the third question I learned that hooking and holding the students' attention with authentic work, giving them clarity of both the detail and the big picture, and connecting the pieces of knowledge they have to the context of the problem they are solving all support these efforts. Conversely, lack of authenticity, clarity and connected knowledge may hinder them. While the data presented in my three papers provide examples of both, the examples I found that demonstrates what is possible have been the main focus. While focusing on possibilities first and limitations second can be said to be an optimistic view, an initial focus on limitations would have made investigating possibilities harder to motivate than the other way around.

This is not to say that I do not acknowledge these limitations, however. The most striking is the barrier that learning to program represents in itself. Students can become so fixated on getting the syntactic details right that they miss out on the big picture entirely. In the last two papers especially, the interviewer had to offer assistance on programming specifics to keep the process running smoothly. We should therefore be cautious about being too ambitious on our students' behalf and consider that they may get stuck in unexpected places. Often, students will need opportunities to catch up and re-learn basic knowledge that is suddenly relevant in a new setting. Without sufficient dots, it is hard to connect them, and doing this catching-up in the middle of an already demanding activity is likely to be cognitively demanding, to say the least.

### 7.4 Implications for Teaching

For teaching, these results have several implications, which I have sorted into four categories.

First, I found that when we integrate computing with mathematics and science, students do not necessarily ignore the mathematical or scientific content and leave everything to the computer. Although they may be tempted to resort to black box thinking when a task is seen as very difficult, the interviewed students frequently and flexibly connected knowledge from both domains in order to make sense of concepts from mathematics or physics. Clarity and access to resources to reason with seems to have been important to make this work.

Second, I have demonstrated that students may be motivated by authentic work in this context and integrating computing into mathematics and science has the potential to offer more such opportunities. I am particularly satisfied with the Taylor expansion tutorial (Section B), because it makes two mathematical

concepts (transforming an infinite interval to a finite one and Taylor expanding a known function) relevant in solving the real-life problem of programming a logarithm function. In other words, I have designed activities that *show* students how a concept is relevant instead of *telling* them that it is. Based on my findings, I would encourage educators to develop other such examples and researchers to validate how these affect students' reasoning.

Third, I have shown that computational concepts may be used as resources for students when they reason mathematically. The best example is perhaps how thinking about how to implement a program helped Sophia re-interpret the numbers in her simple mathematical model in Section I as averages instead of single results. While I cannot claim that involving computing will always have this effect, I believe that when the concepts involved lend themselves to computational reasoning, there is a potential for greater understanding that careful design can unlock.

Last but not least, I have identified a need for students to be supported in translating (as in transferring) their thinking across domains. This goes beyond familiarity and practice with the basics, although I suspect being fluent in both mathematics and computing is beneficial in this regard. Sophia's case was extraordinary in that a single student managed to view the same situation from three different perspectives (mathematics, physics, computing) and connect these. The students in the three cases of Section II faced some obstacles when making these connections, and teaching students to recognise how knowledge from a different domain is relevant is an important understanding for them to take away from the work.

As an example, compare Rita and Lena's initial black box solution with the loop they set up after some support from the interviewer in Section III. For these students, solving an otherwise unsolvable equation by systematic trial and error, in order from few to many terms, was not a simple concept that they could come to easily on their own. If these applications are taught alongside computational concepts like loops, students may make these connections with greater ease. At the same time, I would not recommend that mathematics classes teach or imply that black box thinking is appropriate when a problem is hard or impossible to solve. It may be better to focus more on *how* the calculator or computer is able to help us solve these problems, at least when the concepts involved are seen as relevant to the students.

## 7.5 Implications for Future Research

An important way we could use the data sets of these papers for future research would be to look at examples that are not as exciting as the ones my three papers revolved around. In light of these findings, can we identify the reasons not all the students came to the understandings the students in my three papers did? Are there obstacles to sensemaking, transfer and teaching design in my data that I have not illuminated fully? Where are the thresholds of mathematical and computational knowledge that are required for students to meaningfully take

## 7. Discussion and Conclusions

---

part in these learning activities?

Furthermore, can we use the three tutorials in Section A to collect data more broadly? Having identified some desired understandings and learning goals, it is conceivable that we could design versions of these tutorials where a much larger number of students provide evidence for what they understand, and this could tell us something about how common the examples I provided in this thesis might be.

Among the connections we observed in Paper II, we sometimes saw students pausing halfway between domains, making another half-connection to complete the full connection. Are these halfway points important stepping-stones for students when they are learning to integrate knowledge from different domains? Are they desirable to design for, and how would we do that? How do students progress from these partial connections to complete connections, and how can we support them in doing so?

In the same paper, we noticed an interesting belief being expressed by Gina: that the computer only knows what we explicitly tell it. She seemed surprised to learn that the computer could take advantage of "hidden" or implicit knowledge, for instance of a mathematical nature. I suspect this belief stems from programming instructors' insistence that when we ask the computer to do something, we will need to be very specific, because the computer will take it very literally. It would be interesting to investigate whether this mindset (which may well be beneficial to novice programmers) can be an obstacle when we want students to connect knowledge across domains, such as when we integrate computing in a mathematical context.

In Sections I and III, we found that sometimes students focus too much on *how* a piece of code works that they have difficulties seeing the code's *purpose*. In computer science terms, these students confuse the function's *interface* with its *implementation* (see for instance "Implementing an Interface", 1995). I can imagine conducting teaching experiments that are designed around this distinction, to investigate to what extent thinking in these terms help students look at code from two points of view in the context of integrating computing with mathematics and science.

In Section 6, we briefly noted that some students doing "paper math" on the keyboard in online interviews (due to COVID-19 restrictions) appeared to interpret this work as computational in a way that we never saw in the interviews where students wrote on a whiteboard. This link between the physical medium and students' framing of their work and to what extent it assists or hinders the students in understanding the concepts is another thing that one could investigate further.

It would also be of interest to study how the introduction of computer programming into Norwegian high schools affects students tendency to resort to black box thinking when faced with difficult problems, and what role teaching design plays in this. Will there be systematic differences attributable to teaching that explain why students from some classes are more liable to put blind trust in computational aids than others? In the same way, we might also have a brief window of opportunity to compare current university students, who get a fresh



start in programming, with future students at the same level who have previous experience with programming from school.



# Bibliography

- Anderson, J. M., Tsen, C., Wang, L.-K., Compton, K., & Schulte, J. M. (2009). Performance analysis of decimal floating-point libraries and its impact on decimal hardware and software solutions [ISSN: 1063-6404]. *2009 IEEE International Conference on Computer Design*, 465–471.
- Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. *Digital Experiences in Mathematics Education*, vol. 3no. 2, 115–138.
- Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International Journal of Child-Computer Interaction*, vol. 16, 68–76.
- Biccard, P. (2018). Mathematical sense-making through learner choice [Number: 1]. *Pythagoras*, vol. 39no. 1, 9.
- Billett, S. (2013). Recasting transfer as a socio-personal process of adaptable learning. *Educational Research Review*, vol. 8, 5–13.
- Brown, J., Stillman, G., & Herbert, S. (2004). Can the notion of affordances be of use in the design of a technology enriched mathematics curriculum?
- Buteau, C., Gueudet, G., Muller, E., Mgombelo, J., & Sacristán, A. I. (2020). University students turning computer programming into an instrument for ‘authentic’ mathematical work. *International Journal of Mathematical Education in Science and Technology*, vol. 51no. 7, 1020–1041.
- Buteau, C., Muller, E., Marshall, N., Sacristán, A. I., & Mgombelo, J. (2016). Undergraduate mathematics students appropriating programming as a tool for modelling, simulation, and visualization: A case study. *Digital Experiences in Mathematics Education*, vol. 2no. 2, 142–166.
- Caballero, M. D., Kohlmyer, M. A., & Schatz, M. F. (2012). Implementing and assessing computational modeling in introductory mechanics. *Physical Review Special Topics - Physics Education Research*, vol. 8no. 2, 020106.
- Caglayan, G. (2016). Teaching ideas and activities for classroom: Integrating technology into the pedagogy of integral calculus and the approximation of definite integrals. *International Journal of Mathematical Education in Science and Technology*, vol. 47no. 8, 1261–1279.
- Carlucci Aiello, L. (2016). The multifaceted impact of Ada Lovelace in the digital age. *Artificial Intelligence*, vol. 235, 58–62.
- Chiu, J., & Linn, M. (2011). Knowledge integration and wise engineering. *Journal of Pre-College Engineering Education Research (J-PEER)*, vol. 1no. 1.

- Connor, R., Cutts, Q., & Robertson, J. (2017). Keeping the machinery in computing education. *Communications of the ACM*, vol. 60no. 11, 26–28.
- Costley, J. (2021). How role-taking in a group-work setting affects the relationship between the amount of collaboration and germane cognitive load. *International Journal of Educational Technology in Higher Education*, vol. 18no. 1, 24.
- Cowlshaw, M. (2002). *Densely packed decimal encoding*. Retrieved May 12, 2021, from <http://speleotrove.com/decimal/DPDecimal.html>
- DeJarnette, A. F. (2019). Students' challenges with symbols and diagrams when using a programming environment in mathematics. *Digital Experiences in Mathematics Education*, vol. 5no. 1, 36–58.
- Dimiceli, V. E., Lang, A. S. I. D., & Locke, L. (2010). Teaching calculus with wolfram|alpha [Publisher: Taylor & Francis, Ltd]. *International Journal of Mathematical Education in Science and Technology*, vol. 41no. 8, 1061–1071.
- diSessa, A. A. (2014). *A history of conceptual change research: Threads and fault lines*.
- diSessa, A. A. (1993). Toward an epistemology of physics. *Cognition and Instruction*, vol. 10no. 2, 105–225.
- diSessa, A. A. (2017). Conceptual change in a microcosm: Comparative learning analysis of a learning event [Publisher: Karger Publishers]. *Human Development*, vol. 60no. 1, 1–37.
- du Bolay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices [Publisher: Academic Press]. *International Journal of Man-Machine Studies*, vol. 14no. 3, 237–249.
- Falbel, A. (1991). The computer as a convivial tool. In I. Harel & S. Papert (**typeredactors**), *Constructionism*. Ablex Publishing.
- Ford, M. J. (2012). A dialogic account of sense-making in scientific argumentation and reasoning. *Cognition and Instruction*, vol. 30no. 3, 207–245.
- Forsythe, G. E., Galler, B. A., Hartmanis, J., Perlis, A. J., & Traub, J. F. (1970). Computer science and mathematics. *ACM SIGCSE Bulletin*, vol. 2no. 4, 19–29.
- Gibson, J. J. (1979). *The ecological approach to visual perception* [Google-Books-ID: 8BSLBQAAQBAJ]. Psychology Press.
- Gravemeijer, K., Stephan, M., Julie, C., Lin, F.-L., & Ohtani, M. (2017). What mathematics education may prepare students for the society of the future? *International Journal of Science and Mathematics Education*, vol. 15no. 1, 105–123.
- Greenstein, S. (2018). Designing a microworld for topological equivalence. *Digital Experiences in Mathematics Education*, vol. 4no. 1, 1–19.
- Hammer, D., Elby, A., Scherr, R. E., & Redish, E. F. (2005). Resources, framing, and transfer. In J. Mestre (Ed.), *Transfer of*.
- Hewitt, D. (2016). Designing educational software: The case of grid algebra. *Digital Experiences in Mathematics Education*, vol. 2no. 2, 167–198.

- Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A scoping review of studies on computational thinking in k–12 mathematics classrooms. *Digital Experiences in Mathematics Education*, vol. 4no. 1, 48–69.
- Hohensee, C. (2011). *Backward transfer : How mathematical understanding changes as one builds upon it* (Doctoral dissertation). UC San Diego.
- IEEE standard for floating-point arithmetic [Conference Name: IEEE Std 754-2019 (Revision of IEEE 754-2008)]. (2019). *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 1–84.
- Implementing an interface*. (1995). Retrieved June 12, 2021, from <https://docs.oracle.com/javase/tutorial/java/landI/usinginterface.html>
- Kapon, S. (2017). Unpacking sensemaking. *Science Education*, vol. 101no. 1, 165–198.
- Knuth, E. J., Alibali, M. W., McNeil, N. M., Weinberg, A., & Stephens, A. C. (2011). Middle school students’ understanding of core algebraic concepts: Equivalence & variable. In J. Cai & E. Knuth (Eds.), *Early algebraization: A global dialogue from multiple perspectives* (pp. 259–276). Springer.
- Kudryavtsev, L. D. (2017). *Dirichlet-function - encyclopedia of mathematics*. Retrieved July 1, 2021, from <https://encyclopediaofmath.org/index.php?title=Dirichlet-function>
- Lavy, I. (2006). A case study of different types of arguments emerging from explorations in an interactive computerized environment. *The Journal of Mathematical Behavior*, vol. 25no. 2, 153–169.
- Lobato, J. (2003). How design experiments can inform a rethinking of transfer and vice versa [Publisher: American Educational Research Association]. *Educational Researcher*, vol. 32no. 1, 17–20.
- Lobato, J. (2008, January 1). When students don’t apply the knowledge you think they have, rethink your assumptions about transfer.
- Lobato, J. (2012). The actor-oriented transfer perspective and its contributions to educational research and practice. *Educational Psychologist*, vol. 47no. 3, 232–247.
- Lobato, J., & Siebert, D. (2002). Quantitative reasoning in a reconceived view of transfer. *The Journal of Mathematical Behavior*, vol. 21no. 1, 87–116.
- Lockwood, E., & De Chenne, A. (2020). Enriching students’ combinatorial reasoning through the use of loops and conditional statements in python. *International Journal of Research in Undergraduate Mathematics Education*, vol. 6no. 3, 303–346.
- Lockwood, E., & Mørken, K. (2021). A call for research that explores relationships between computing and mathematical thinking and activity in RUME. *International Journal of Research in Undergraduate Mathematics Education*.
- Maloney, D. (2015). Teaching critical thinking: Sense-making, explanations, language, and habits. *The Physics Teacher*, vol. 53no. 7, 409–411.
- Malthe-Sørenssen, A., Hjorth-Jensen, M., Langtangen, H. P., & Mørken, K. (2015). Integrating computation in the teaching of physics. *UNIPED*, 9.
- McPeck, J. E. (2016, September 13). *Critical thinking and education* [Google-Books-ID: E1IPDQAAQBAJ]. Routledge.

- Muller, J.-M., Brisebarre, N., de Dinechin, F., Jeannerod, C.-P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., & Torres, S. (2010). Floating-point formats and environment. In J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, & S. Torres (Eds.), *Handbook of floating-point arithmetic* (pp. 55–116). Birkhäuser.
- Mørken, K. (n.d.). *MAT-INF1100 – modelling and computations - universitetet i oslo*. Retrieved January 29, 2021, from <https://www.uio.no/studier/emner/matnat/math/MAT-INF1100/index-eng.html>
- Mørken, K. (2017, September). Numerical algorithms and digital representation.
- Nederbragt, A. J. (n.d.). *BIOS1100 – introduction to computational models for biosciences - universitetet i oslo*. Retrieved February 4, 2021, from <https://www.uio.no/studier/emner/matnat/ibv/BIOS1100/index-eng.html>
- Nowell, L. S., Norris, J. M., White, D. E., & Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria [Publisher: SAGE Publications Inc]. *International Journal of Qualitative Methods*, vol. 16no. 1, 1609406917733847.
- Odden, T. O. B., & Russ, R. S. (2018). Defining sensemaking: Bringing clarity to a fragmented theoretical construct. *Science Education*, vol. 0no. 0.
- Odden, T. O. B., & Russ, R. S. (2019). Vexing questions that sustain sensemaking. *International Journal of Science Education*, vol. 41no. 8, 1052–1070.
- Olsson, J. (2019). Relations between task design and students' utilization of GeoGebra. *Digital Experiences in Mathematics Education*, vol. 5no. 3, 223–251.
- Papert, S. A. (1993, August 4). *Mindstorms: Children, computers, and powerful ideas* (2 edition). Basic Books.
- Pena-Marin, J., & Yan, D. (2021). Reliance on numerical precision: Compatibility between accuracy versus efficiency goals and numerical precision level influence attribute weighting in two-stage decisions. *Journal of Consumer Psychology*, vol. 31no. 1, 22–36.
- Penprase, B. E. (2020). Theories of teaching and learning. In B. E. Penprase (Ed.), *STEM education for the 21st century* (pp. 35–50). Springer International Publishing.
- Ramler, I. P., & Chapman, J. L. (2011). Introducing statistical research to undergraduate mathematical statistics students using the guitar hero video game series. *Journal of Statistics Education*, vol. 19no. 3, null.
- Reddy, M. V. B., & Panacharoensawad, B. (2017). Students problem-solving difficulties and implications in physics: An empirical study on influencing factors [Publisher: IISTE]. *Journal of Education and Practice*, vol. 8no. 14, 59–62.
- Ryghaug, M., Holtan Sørensen, K., & Næss, R. (2011). Making sense of global warming: Norwegians appropriating knowledge of anthropogenic climate change [Publisher: SAGE Publications Ltd]. *Public Understanding of Science*, vol. 20no. 6, 778–795.
- Sand, O. P. (2016). *Massive neutrinos and spherical collapse in LCDM and DGP gravity* (Doctoral dissertation).

- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, vol. 22, 142–158.
- Sinclair, N., & Patterson, M. (2018). The dynamic geometrisation of computer programming. *Mathematical Thinking and Learning*, vol. 20no. 1, 54–74.
- Stormo, A. (2009). Integrering av numeriske beregninger i grunnleggende fysikkurs (in norwegian) [Accepted: 2014-12-19T13:16:21Z Publisher: Norges teknisk-naturvitenskapelige universitet, Fakultet for naturvitenskap og teknologi, Institutt for fysikk].
- Strike, K. A., & Posner, G. J. (1982). Conceptual change and science teaching. *European Journal of Science Education*, vol. 4no. 3, 231–240.
- Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking [event-place: Koli, Finland]. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 120–129.
- UDIR. (2020). *Hva er nytt i matematikk?* Retrieved May 12, 2021, from <https://www.udir.no/laring-og-trivsel/lareplanverket/fagspesifikk-stotte/nytt-i-fagene/hva-er-nytt-i-matematikk/>
- van Someren, M. W., Barnard, Y., & Sandberg, J. (1994). *The think aloud method: A practical guide to modelling cognitive processes*. Academic Press, Inc.
- Wagner, J. F. (2010). A transfer-in-pieces consideration of the perception of structure in the transfer of learning. *Journal of the Learning Sciences*, vol. 19no. 4, 443–479.
- Watters, D. J., & Watters, J. J. (2006). Student understanding of pH: “i don’t know what the log actually is, i only know where the button is on my calculator”. *Biochemistry and Molecular Biology Education*, vol. 34no. 4, 278–284.
- Weinreich, D. M., Sivapalasingam, S., Norton, T., Ali, S., Gao, H., Bhore, R., Musser, B. J., Soo, Y., Rofail, D., Im, J., Perry, C., Pan, C., Hosain, R., Mahmood, A., Davis, J. D., Turner, K. C., Hooper, A. T., Hamilton, J. D., Baum, A., . . . Yancopoulos, G. D. (2021). REGN-COV2, a neutralizing antibody cocktail, in outpatients with covid-19. *New England Journal of Medicine*, vol. 384no. 3, 238–251.
- Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, vol. 25no. 1, 127–147.
- Wiggins, G., & McTighe, J. (2005, January 1). *Understanding by design* (2nd Expanded edition). Assn. for Supervision & Curriculum Development.
- Wright, G., Rich, P., & Lee, R. (2013). The influence of teaching programming on learning mathematics, 4612–4615.





# Appendices



# Appendix A

## Tutorial 1: Rounding errors

Here, I present the final versions of the tutorials we made for the course *MAT-INF1100: Modelling and Computations*. We expect these to also be useful outside of the course they were designed for and the University of Oslo context.

### A.1 Final Version of Tutorial

#### A.1.1 Introduction

In this tutorial we will look at the use of a Python program to do a simple calculation. The program adds probabilities, and these should always sum to exactly 1 (100%). But if these sums are not always 1, it could be a problem. In fact, if they are far from 1, it could be a *big* problem.

Your job is to test the program and fix it if need be. The future of the statistics community depends on the work that you do here.

All the exercises are designed to be done in groups of at least 2 students. The goal is that everyone in your group understands what is happening and why. We encourage you to check in with your TA when you have questions, feel stuck or just want to test the soundness of your reasoning.

#### A.1.2 Exercises

##### A.1.2.1 Exercise 1

For each statement, indicate if you agree or disagree:

- Rounding errors are always small.
- You either do computing or mathematics, you cannot do both at once.
- A computer can only do what we explicitly tell it to do, in clear terms.
- Mathematical proofs and computer programs are unrelated concepts.

You do not have to reach a consensus on these, different opinions are fine, but in any event try to justify your answers to the rest of the group.

##### A.1.2.2 Exercise 2

Look at the provided program `sum.py` (Code Sample A.1). Feel free to run it to see what the output looks like. Explain briefly in your own words what this program is doing.

### A.1.2.3 Exercise 3

Do the same calculation on paper. What result do you get? Would you expect this program to give the same result no matter what the starting value is? Can you prove this?

### A.1.2.4 Exercise 4

What happens if you change the starting value to 0.5? Explain the result.

### A.1.2.5 Exercise 5

What happens if you change the starting value to 0.1? Explain the result.

### A.1.2.6 Exercise 6

What is the binary representation of 0.1? How could this possibly be related to what happened in exercise 5?

### A.1.2.7 Exercise 7

Fix the program so that it behaves as intended (see exercise 3) and test it. Explain what you did and how it solved the problem from exercise 5. You are allowed to be creative here.

### A.1.2.8 Exercise 8

For each statement, indicate if you agree or disagree, just like in exercise 1:

- Rounding errors are always small.
- You either do computing or mathematics, you cannot do both at once.
- A computer can only do what we explicitly tell it to do.
- Mathematical proofs and computer programs are unrelated concepts.

If you changed your mind (as a group or individually) on any of these, what did you experience that made you see things differently? How can this be useful to you in the future?

## A.2 Code for Tutorial 1

```
x0 = 0.91          # initial value
n = 9             # number of steps

step = (1 - x0)/n # step length

i = 0            # step counter
sum = x0
print()
print("Step", i, ":", sum) # step 0 er just the initial value (
    no steps taken yet)
while sum < 1.0:
    i = i + 1
    sum = sum + step # cumulative sum (i.e. the sum so far)
    print("Step", i, ": +", step, "=", sum)

print()
print("Result :", sum, "after", i, "steps")
print("Expected: 1.0 after", n, "steps")
```

Code Sample A.1: The code given to students for tutorial 1.



# Appendix B

## Tutorial 2: Taylor Expansions

### B.1 Final Version of Tutorial

#### B.1.1 Introduction: Your very own log function

Have you ever wondered *how* computers and calculators know how to calculate logarithms? And what is the point of Taylor polynomials when the function we use is known already?

In this tutorial we will investigate these questions.

A computer excels at working with the four basic calculations: adding, subtracting, multiplying, and dividing. That makes it easy for the computer to work with polynomials. But how do you go from there to more advanced functions like logarithms? You are about to find out because we are going to make one.

The main point of these exercises is that everyone in the group should:

- understand how the computer calculates logarithms,
- understand the role the exponential representation of floating-point numbers plays in this, and
- understand at least one practical application of Taylor polynomials for known functions.

Make sure to ask questions and explain to each other along the way so the whole group ends up with these understandings. This is much more important than getting the right answer.

#### B.1.2 Exercises

##### B.1.2.1 Exercise 1

In these exercises we will work with the natural logarithm  $\ln(x)$  or `log(x)` as it is called in Python.

Take a look at the file `taylorlog.py` (Code Sample B.1). It contains the following functions:

Note the meaning of the following variable names:

Your first task is to complete the `taylorterm` function, so it returns the *i*th term of the Taylor polynomial. Explain how you go from the general formula to

## B. Tutorial 2: Taylor Expansions

---

Table B.1: Functions for Tutorial 2.

Function name	Purpose	Completed?
<code>taylorterm</code>	Calculates one term of the Taylor polynomial for the logarithm	No (exercise 1)
<code>taylor</code>	Calculates the entire Taylor polynomial for the logarithm	Yes
<code>remainder</code>	Calculates the absolute remainder of the Taylor polynomial	No (exercise 1)
<code>taylorlog</code>	A smarter way to calculate the natural logarithm that makes use of the Taylor polynomial	No (exercise 3)

Table B.2: Variables for Tutorial 2.

Variable	Meaning
<code>i</code>	Any term in the Taylor polynomial
<code>n</code>	Maximal number of terms in the Taylor polynomial
<code>a</code>	The point we use to make the Taylor expansion (i.e., where it is most accurate)
<code>lna</code>	The natural logarithm of <code>a</code>
<code>x</code>	Any point where we want to calculate the logarithm using the Taylor expansion
<code>xi</code> ( $\xi$ )	Some unknown point somewhere between <code>a</code> and <code>x</code> that we use to calculate the remainder

the one specific to the logarithm. Note that we use `lna` as the input parameter instead of `a` for practical reasons <sup>1</sup>.

### B.1.2.2 Exercise 2

Your next task is to also complete the remainder function, so it returns the correct absolute remainder for the natural logarithm with `n` terms:

$$\left| \frac{(x - a)^{n+1}}{(n + 1)!} f^{(n+1)}(\xi) \right|$$

Again, explain how you go from the general formula to the one specific to the logarithm.

---

<sup>1</sup>This means that instead of changing the value of `a` directly, we need to change `lna` instead. If we need the value of `a`, we can get it by using `a = exp(lna)`. We do this because term zero in the Taylor polynomial is simply equal to  $\ln a$ . We can't calculate this value without a pre-existing log function, but there is nothing wrong with simply *choosing* what  $\ln a$  should be, and then calculating `a` if we need to know what it is.



A problem here is that you do not really know the value of  $x_i$ , only that it is somewhere between  $x$  and  $a$  (both are positive for the logarithm, but we do not know which one is larger). Work around this issue by using the worst-case value (the one that makes the remainder as *large* as possible), so we get an upper bound for the error.

Why do we want this estimate of the error to be as large as possible, when it will probably be larger than the actual error?

### B.1.2.3 Exercise 3

Test your code by using the program `taylor_test.py` (Code Sample B.2).

With the default values `lna = -0.15`, `n = 10` it should produce the following plot of the absolute remainder. The dotted line in the plot below is a limit of  $10^{-10}$  which we accept as accurate enough in practice for this tutorial.

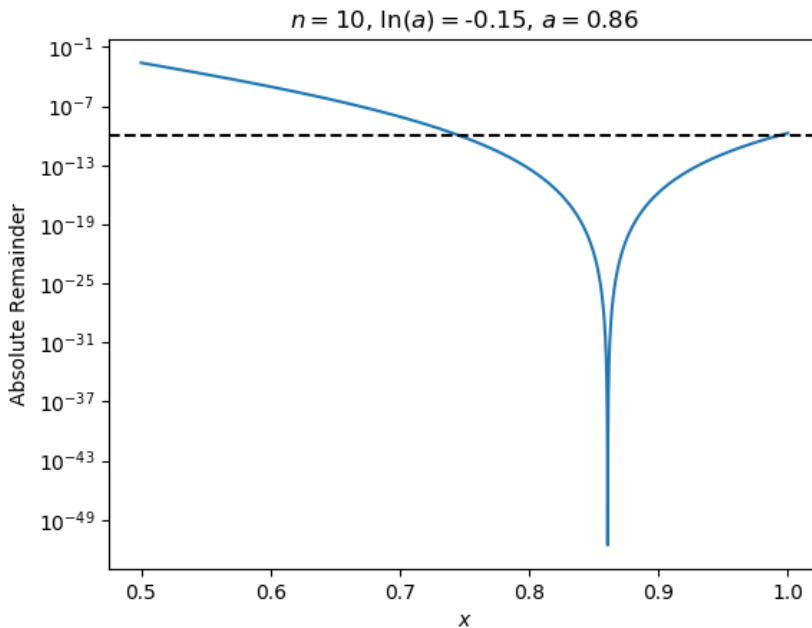


Figure B.1: Test plot for self-assessment in Tutorial 2.

If your code produces a different plot than this, you will need to do some debugging and correct it before moving on.

### B.1.2.4 Exercise 4

Experiment with changing the values of `lna` and `n` in `taylor_test.py` until you can explain what happens to the plot of the remainder when you change each of these: what is the effect of changing the point `a`? What is the effect of changing the number of terms `n`?

### B.1.2.5 Exercise 5

It may not come as a surprise that Taylor polynomials are most accurate close to the point `a`, and much less accurate far away from that point. Since we can only pick one point at a time, this makes it very hard to use Taylor polynomials to give us a logarithm function for *all* positive real numbers.

To work around this problem, we will exploit the fact that on the computer, real numbers are represented as floating point numbers on the standard form:

$$x = \text{mantissa} * 2^{**\text{exponent}}$$

Note that `mantissa` is a number between 0.5 and 1 and `exponent` is a unique integer. What happens, mathematically, to the right-hand side when you take the logarithm of both sides?

Look at your answer and consider how it might be an improvement to use `taylor(mantissa)` instead of just `taylor(x)`. Explain which one of these it is easiest to get accurate values from. (Remember that `x` can be any positive number and that `mantissa`'s range of possible values are very limited.)

### B.1.2.6 Exercise 6

Use the expression you found in exercise 5 to finish the function `taylorlog` in `taylorlog.py`. As you can see, Python already has a way to get `mantissa` and `exponent` from the number `x`.

Remember that the `taylor` function can be used when you need a logarithm instead of `log` – we are trying to build our own logarithm from scratch here, so we don't want to cheat. There is one exception to this rule: you can use the known value of the logarithm of 2, which is given in the code.

### B.1.2.7 Exercise 7

To test if your `taylorlog` function works, run the program `log_test.py` (Code Sample B.3) with the standard values `lna = 2` and `n = 10`. If it worked, you should see this result (or at least something very close to it):

We see that the accuracy is far from perfect. Don't worry, though; in the next exercise we'll improve the accuracy.

If you get different results, check your code and fix any errors you find.

```

x = 0.001
---
log:          -6.907755278982137
taylorlog:    -6.907748282562315
relative error: 1.0128355072806373e-06

x = 0.1
---
log:          -2.3025850929940455
taylorlog:    -2.3025850929940246
relative error: 9.0646781856009e-15

x = 10
---
log:          2.302585092994046
taylorlog:    2.3025851719118426
relative error: 3.427356364899646e-08

x = 1000
---
log:          6.907755278982137
taylorlog:    6.9077552789608685
relative error: 3.078904735211335e-12

ln(a) = -0.15
a      = 0.8607079764250578
n      = 10

```

Figure B.2: Output of test program of log function.

### B.1.2.8 Exercise 8

Since we cannot use any built-in log function for this, use the `taylor` function to find a value of `lna` that gives you a really low relative remainder in the *entire* interval 0.5 and 1.

The problem is, we only have machine accurate values of two logarithms: `ln2` and `ln3`. These are defined in `taylor_test.py`, and you can use that program to test your values (remember that  $10^{-10}$  is considered close enough in practice for the remainder in is tutorial).

### B.1.2.9 Exercise 9

Now that you have some values that work well enough, investigate mathematically how few terms `n` you can get away with and still have a remainder that is smaller than  $10^{-10}$ .

- You will need to pick a value of `x`. Which one is the best choice when we want a small remainder for *all* `x` between 0.5 and 1?
- Remember to pick the worst-case value of `xi`.
- You should end up with an expression where `n` is the only unknown quantity.

Try to find the value of `n` analytically. If you cannot, is there a way for Python to test many different values of `n` and pick the smallest one that gives a satisfactory remainder?

When you have an answer, run `taylor_test.py` to verify that the remainder is not too large anywhere for this number of terms.

### B.1.2.10 Exercise 10

Once you are happy with the result, use `log_test.py` to test your very own logarithm function from Exercise 7 using the best values of `lna` and `n` that you found.

- If the relative error is larger than 0 for some of these examples, adjust `n` some more until the error is 0 for all of them.

### B.1.2.11 Exercise 11 (Optional)

Consider the expression for the remainder at the two endpoints in Table B.3. Explain the difference between the remainders at the endpoints in Figure B.3.

Table B.3: Remainder values for Exercise 11.

$x$	$a$	$\xi$	Remainder
1	0.75	0.75	$\frac{1}{n+1} \left  \left( \frac{x-a}{\xi} \right)^{n+1} \right $
0.5	0.75	0.5	

### B.1.2.12 Exercise 12

Finally, sum up in your own words, what we have done, especially:

- How the computer calculates logarithms<sup>2</sup>,
- How we checked that the calculations were accurate (enough),
- Where the Taylor expansion was useful to us, even though we already knew the function it was approximating,
- How we changed where the Taylor expansion needed to be accurate from all positive numbers to a small interval between 0.5 and 1, and
- How we used both mathematics and programming to identify the smallest possible number of terms.

Your goal here is that *all* of you understand what you did and why. You are encouraged to ask a TA if you have problems explaining or understanding something, or just want to check that you have a good enough understanding of what you did.

---

<sup>2</sup>In practice, even faster approximations with less terms than the Taylor polynomial are often used, but the principle is the same.

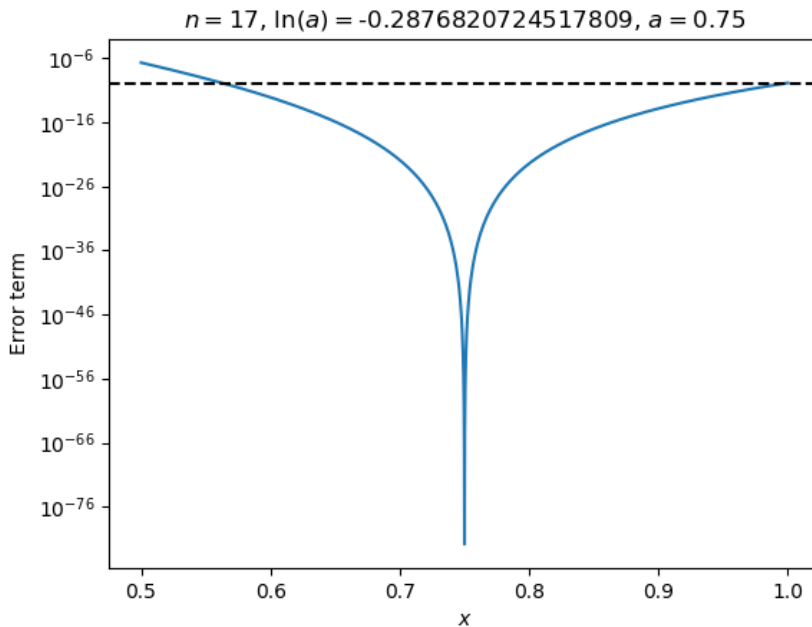


Figure B.3: Asymmetric remainder to be explained by the students.

## B.2 Code for Tutorial 2

```

from numpy import exp
from math import frexp

# Exercise 1: Calculate the i'th term of the Taylor polynomial
# We have already taken care of term 0, which is simply ln(a)
def taylorterm(x, lna, i):
    if i == 0:
        return lna
    else:
        a = exp(lna)
        # TODO: calculate the term for i > 0
        return 0

# Adds together all the different Taylor terms up to and
# including n
def taylor(x, lna, n):
    sum = 0.0
    for i in range(n + 1):

```

## B. Tutorial 2: Taylor Expansions

---

```
        sum += taylorterm(x, lna, i)
    return sum

# Exercise 2: Calculate the remainder of the Taylor polynomial
def remainder(x, lna, n):
    a = exp(lna)
    # TODO: calculate the remainder
    return 0

# Exercise 6: Your very own logarithm function!
def taylorlog(x, lna, n):
    # get the two parts of the number on the form they are
    # stored in memory
    mantissa, exponent = frexp(x)
    ln2 = 0.6931471805599453 # ln(2) with machine accuracy (16
    digits)
    # TODO: Calculate ln(x) by using what you found in Exercise
    # 5
    # NOTE: Do not use log(...) for any of this.
    #       We should use taylor(...) for all out logarithm
    # needs.
    return 0
```

Code Sample B.1: The file `taylorlog.py` where students make their own log function.

```
import matplotlib.pyplot as plt
from numpy import abs, linspace, exp
from taylorlog import taylor, remainder

# These constants might be helpful at some point
ln2 = 0.6931471805599453 # ln(2) with machine accuracy (16
    digits)
ln3 = 1.0986122886681097 # ln(3) with machine accuracy (16
    digits)

# These values affect the Taylor polynomial, and can be changed
:
lna = -0.15
n = 10

# There should be no need to change anything below this point:
xmin = 0.5
xmax = 1
xs = linspace(xmin, xmax, 10000)

remainders = []
for x in xs:
    rem = remainder(x, lna, n)
    remainders.append(rem)

plt.plot(xs, remainders)
plt.axhline(10**-10, linestyle="--", color="black")
plt.xlabel("x")
plt.ylabel("Absolute Remainder")
plt.yscale("log")
plt.title("n=" + str(n) + ", \ln{(a)}=" + str(lna) + ", a=" +
    str(round(exp(lna), 2)))
plt.show()
```

Code Sample B.2: The file `taylor_test.py` that plots the remainder.

## B. Tutorial 2: Taylor Expansions

---

```
from numpy import log, exp
from taylorlog_solution import taylorlog

# These constants might be helpful at some point
ln2 = 0.6931471805599453 # ln(2) with machine accuracy (16
    digits)
ln3 = 1.0986122886681097 # ln(3) with machine accuracy (16
    digits)

# These values affect the Taylor polynomial, and can be changed
:
lna = -0.15
n = 10

# There should be no need to change anything below this point:
xs = [0.001, 0.1, 10, 1000]
for x in xs:
    logx = log(x)
    taylorlogx = taylorlog(x, lna, n)
    relerr = abs((taylorlogx - logx) / logx)
    print()
    print("x =", x)
    print("---")
    print("log:          ", logx)
    print("taylorlog:       ", taylorlogx)
    print("relative error:", relerr)
print()
print("ln(a) =", lna)
print("a      =", exp(lna))
print("n      =", n)
```

Code Sample B.3: The file `log_test.py` that tests the students' log function.



# Appendix C

## Tutorial 3: Numerical Integration

### C.1 Final Version of Tutorial

#### C.1.1 Introduction

In numerical differentiation it is important that the step size is not too short (mathematical error) nor too long (rounding error). Does the same apply to numerical integration? Why/why not?

In this tutorial, we will work with a function that is impossible to integrate by hand, for which we know the values of the integral anyway. It is none other than the *normal distribution function*:

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

For simplicity, we set the mean  $\mu = 0$  and the standard deviation  $\sigma = 1$ , so that we get:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Integrating this from -10 to 10, will definitely give us 1 as the answer (because the odds that something is as far as 10 standard deviations from the mean is much, much smaller than the rounding error on the computer<sup>1</sup>):

$$\int_{-10}^{10} f(x) dx = 1$$

Work together and focus on everyone understanding what is going on. Ask a TA for help if you have trouble convincing each other or want to check that you're on the right track.

#### C.1.2 Exercises

##### C.1.2.1 Exercise 1: The midpoint method

In the module `integrals.py` that you have been given (Code Sample C.1), we have done the following for you:

- Defined the function `f` that represents  $f(x)$  in Python
- Defined a `stepsize` function that you can use to calculate the step size.

---

<sup>1</sup>Try `print(f(-10))` and `print(f(10))` in `integrals.py` if you are curious.

## C. Tutorial 3: Numerical Integration

---

The first thing that is missing is a function `area(x, h)` that calculates the area under the curve between `x` and `x + h` using the midpoint method<sup>2</sup>.

Test the function and check that `area(0,1)` gives the result `0.3520653267642995`.

### C.1.2.2 Exercise 2: The entire integral

The next thing that is missing is a function `integral(a, b, n)` that calculates the entire integral from `a` to `b` with `n` steps.

Once you are happy with your function for numerical integration, test that you get the value `0.9856162386389232` (or something very close to it) for the integral with `a = -10` and `b = 10` using 10 steps.

### C.1.2.3 Exercise 3: The optimal number of steps

The file `plots.py` (Code Sample C.2) uses the functions in `integrals.py` to plot how accurate the integration is with different step sizes. It plots the relative error of the integration as a function of `n` and also prints out all the results to the terminal.

What is the minimal number of steps needed for a perfect result when integrating from -10 to 10?

What is the minimal number of steps needed for a perfect result when integrating from -100 to 100?

Based on the previous two answers, what would you expect to be the answer when integrating from -1000 to 1000?

Test your guess for -1000 to 1000. What happens to the running time of the program as the number of steps grows large?

### C.1.2.4 Exercise 4: What is going on?

Compare the two plots of the numerical integrals from -10 to 10 (left) and -100 to 100 (right). The number of steps used are 28 and 270, respectively. The red dots are the midpoints used to calculate the area under the curve.

Why do you think that we need roughly 10 times as many points when the integration limits are 10 times larger?

Would you expect the same relation to hold for the integral from -1 to 1? Why/why not?

### C.1.2.5 Exercise 5: Many steps!

You may have noticed that the more steps we use, the fewer of the integrals get relative error zero. If we use a logarithmic x-axis for the first integral (from -10 to 10), we find that the relative error seems to increase when `n` grows very large.

Why is the error so large for small `n`?

---

<sup>2</sup>Not Euler's midpoint method for differential equations, but the related method with the same name used for definite integrals.

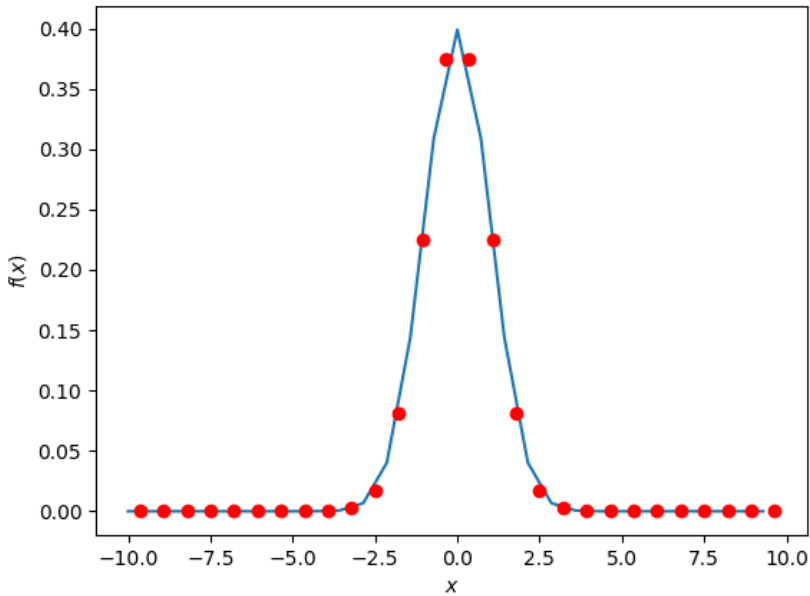


Figure C.1: Normal distribution function between -10 and 10 standard deviations.

Why is the error steadily increasing for large  $n$ ? (Hint: With very many steps, we add very many numbers, some rather large (near  $x = 0$ ) and many of them very close to zero.)

Based on this and the running time you observed for the program, what should one think about when choosing a step size for numerical integration?

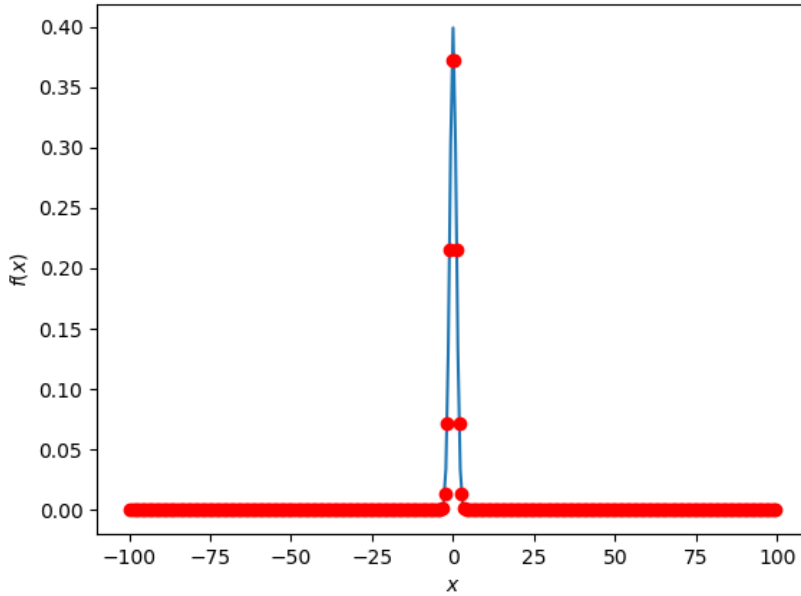


Figure C.2: Normal distribution function between -100 and 100 standard deviations.

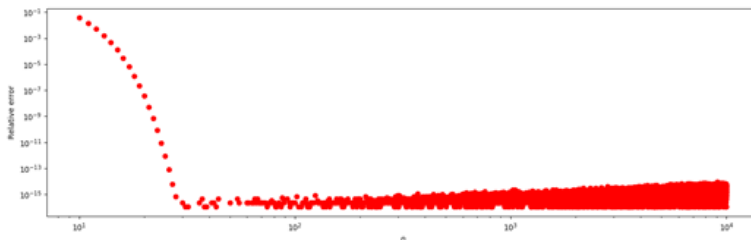


Figure C.3: Relative error scaling of the normal distribution function with increasing number of steps between -10 and 10 standard deviations.

## C.2 Code for Tutorial 3

```
from numpy import exp, pi

# normal distribution with standard values for mu and sigma
def f(x, mu=0, sigma=1):
    return exp(-(x-mu)**2/(2*sigma**2))/((2*pi*sigma**2)**(1/2)
    )

def area(x, h):
    # TODO: Calculate the area under the curve between x and x+
    h (exercise 1)
    return 0

# returns the step size for n steps between x=a and x=b
def stepsize(a, b, n):
    return (b - a)/n # n is the number of steps, not the number
    of points

def integral(a, b, n):
    h = stepsize(a, b, n)
    # TODO: Calculate the integral between a and b using n
    steps (exercise 2)
    return 0
```

Code Sample C.1: The file `integrals.py` where the students integrate the normal distribution.

### C. Tutorial 3: Numerical Integration

---

```
from numpy import arange
import matplotlib.pyplot as plt
from integrals import integral

# plot parameters (exercise 3)
nmin = 1                # smallest value of n
nmax = 100              # largest value of n
nstep = 1               # plot every ...th value of n (
                        # skipping some will speed up the program)

# integration limits (exercise 3)
a = -10
b = 10

ns = arange(nmin, (nmax + 1), nstep)
relerrs = []

for n in ns:
    relerr = abs(integral(a, b, n) - 1)
    relerrs.append(relerr)
    print("n =", n, "/", nmax, "--> rel.err. =", relerr)

plt.plot(ns, relerrs, "o")
plt.xlabel("Number of steps (n)")
plt.ylabel("Relative error")
plt.yscale("log")
filename = "a" + str(a) + "b" + str(b) + "nmin" + str(nmin) + " "
           "nmax" + str(nmax) + "nstep" + str(nstep) + ".png"
plt.savefig(filename)
plt.show()
```

Code Sample C.2: The file `plots.py` that generates plots of the relative error.