Programming in Mathematics Education: An Intermediary between the Real-World and the Mathematical Model

Andreas Brandsæter[1,2][0000−0001−5142−854X]

[1] Department of Science and Mathematics, Volda University College, Norway
andreas.brandsaeter@hivolda.no
[2] Department of ICT and Natural Sciences, NTNU – Norwegian University of Science and Technology, Norway
andreas.brandsater@ntnu.no

Abstract. In realistic mathematics education students expand their common sense through guided reinvention, aiming to prevent the dichotomy between what the students experience as real and the associated mathematical model. The mathematics curriculum is, however, often comprehensive and ambitious, and many students will find it challenging to develop the required skills and knowledge. In order to solve the complicated problems required at exams, students and teachers will yield to memorizing formulas and procedures, letting large parts of mathematics education reside in the formal mathematical model. In this paper we propose to utilize programming as an intermediary between the real-world problems and formal mathematics, not merely as a tool to solve the problems or find approximate solutions, but to increase understanding and to guide the student's mathematical reinvention.

Keywords: Realistic Mathematics Education · ICT in Education · Programming in Mathematics · Programming for Relational Understanding

1 Introduction

Skills and knowledge about digital tools, algorithmic thinking and programming are recognized as vital in many subjects, including science, technology, engineering, and mathematics [14]. Programming is introduced in school curriculum on a national or regional level in many countries, aiming to develop the student's analytical abilities and problem solving skills necessary to learn, work and live in the society of the future [19]. Skills in computational thinking helps the students approach problems with a computational perspective, adapting their thinking processes to align with the algorithms of a computer [1]. Some schools include programming in the curriculum as a separate subject or study discipline, while others include it as part of other subjects, often mathematics. In addition to programming, various digital tools and games can be utilized in mathematics education such as for example the game Dragonbox which is used to teach equations [25], spreadsheets such as Microsoft Excel which is used to teach diagrams

and graphs, how to sort and structure information, budgeting and accounting [24], GeoGebra which is used as a graphic calculator including CAS and a dynamic tool for drawing geometric figures in the plane. Various programming languages, both text based and block-based, can be applied for teaching various topics in mathematics and solving problems.

The value of programming and computational thinking in school curriculum has been argued for by many [28]. However, this paper proposes that programming can be used as an intermediary for understanding classical school mathematics (such as calculus and geometry). We argue that programming has the potential to facilitate new connections between formal mathematics and the real world [9], contributing to bridge the gap between reality and the mathematical model. Programming is a powerful tool to derive the results of many mathematical problems, such as applying numerical methods for computing quantities like the derivative and the integral, but the computational perspective can also be useful for understanding and comprehending the fundamental concepts.

In the following section, we first discuss the gap that can arise between the island of formal mathematics and the mainland of real human experiences [18] affecting both motivation and mathematics performance. We explain how programming can be used as an intermediary to bridge or diminish this gap. The use of simulators and simulation for teaching purposes are discussed in Section 3. We present an example of a mathematical simulator which calculates compound interest by "walking through" a scenario one year at the time. We also provide an example to illustrate how a coin-toss simulator can facilitate the reinvention of the law of large numbers. Through repeated simulations, the students can experience and observe how the results develop when the number of iterations increases. In Section 4, challenges related to how students deal with infinity and the concept of limits are also discussed, and we provide examples of simple algorithms to calculate definite integrals. In Section 5, we explain how mathematical objects can have different representations, and argue that an algorithmic expression can be a useful supplement to the other more common representations such as graph, table and formula. Finally, challenges and limitations with programming in mathematics education are discussed in Section 6, and concluding remarks are offered in Section 7.

2 The Island Problem

Kaput [18] emphasizes the gap between the island of formal mathematics and the mainland of real human experiences (see Figure 1) and discusses how authentic experiences can be linked with formal representations. The translations between the student's real-world environment and the mathematical model environment are often non-trivial, and challenges associated with this translation is perhaps a contributing factor to a large part of mathematics education resides in the formal mathematical model. Education focuses on manipulations of formulas failing to convey the basic concepts which are non-trivial [6,5]. Students end up being able

to solve complicated mixed differentiation problems, and at the same time being unable to apply differentiation to simple real-world situations.

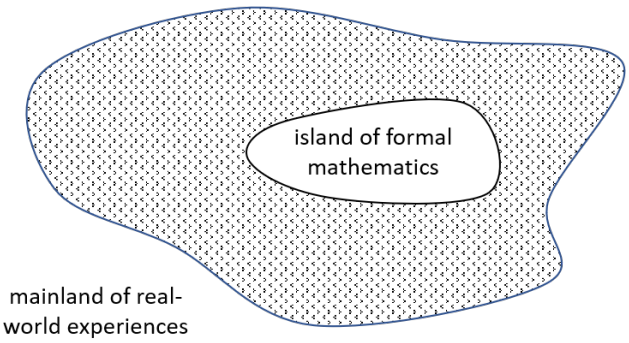


Fig. 1: Kaput [18] emphasizes the gap between the island of formal mathematics and the mainland of real human experiences.

Realistic mathematics education (RME) [10] aims to let formal mathematics emerge through mathematical activity, exposing students to problems which are experimentally real to them [12]. This includes pure mathematical problems as long as the problem offer the student a context making it experimentally real for the student. Through mathematical activity, students can expand their common sense and reinvent mathematical theory. Common sense is (in a sense) the primordial certainty, Freudenthal [10, p. 6] explains. But one should be careful not to be deceived by it. Based on a few datapoints, humans very quickly find patterns which are useful and prove to be correct in some cases, but does not hold generally [17].

To bridge or diminish the gap between everyday experiences and formal mathematics, we propose to use a programming environment as an intermediary. The assumption is that a translation from the real-world environment to a programming environment can be easier to implement and comprehend. When the student is able to formulate and solve the problem or find an approximate solution in the programming environment, it can be translated and represented as a mathematical model.

A sketch of the proposed approach is shown in Figure 2. The figure is partly adapted from Wubbels et al. [29], which includes both the real-world and the mathematical model environment. Wubbels' model illustrates the mathematical inquiry process where a real-world problem is translated to a mathematical model where it is solved using mathematical techniques. The solution is then translated back to the real-world environment, and the student reflects on merits and restrictions associated with the problem [29]. We add a programming environment to this model as a proposed intermediary in the translation be-

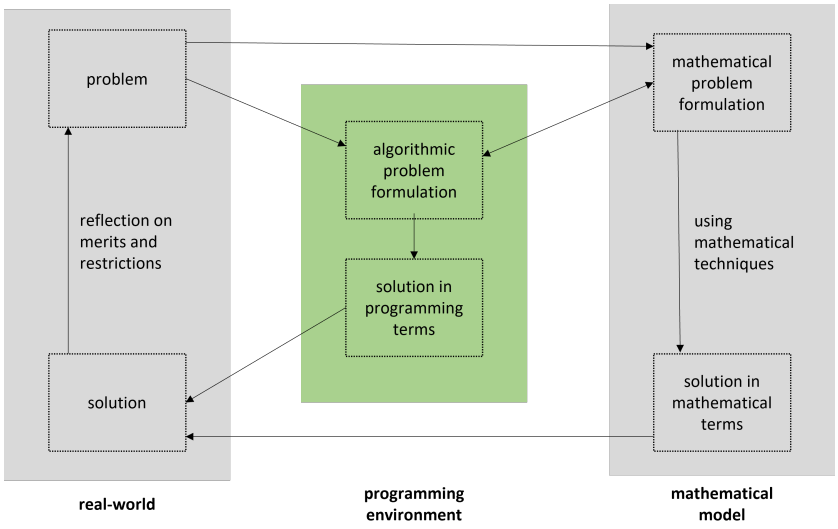


Fig. 2: A model of problem solving which encompasses the translation of a real-world problem to a mathematical formulation, with programming as a potential intermediary. The diagram is adapted from Wubbels et al. [29] which does not include the programming environment.

tween reality and the mathematical model. In the following, we illustrate the presented model by presenting a set of classical mathematics problems, and show how programming can be utilized to solve these problems by repeatedly performing simple operations.

3 Simulation and Simulators

The use of high fidelity simulators for training and education is well-established in many areas such as for example driving or surgery. But can we benefit from utilizing simulators in mathematics education? Driving or surgery simulators offers realistic replication of the real-world environment [4]. Here, students face real-world-like problems which they solve in a real-world-like environment. As long as the simulators are sufficiently realistic, the transfer of learning between the simulator environment and real-world is straightforward. In mathematics education, however, the translation is not always apparent.

3.1 Example: Computer-Simulated Vehicles

Kaput [18] investigates how computer-simulated vehicles with velocity and/or position versus time displays can be utilized to link the phenomenological everyday experience of motion in a vehicle to more formal representations. He

explains how the students can utilize the computer-simulated vehicles and their associated graphs to answer questions regarding distance measurement and area estimation. Generalizing the acquired knowledge to answer other problems such as comparing the motion graphs to that of pay raises [18] is perhaps possible, but far from trivial.

3.2 Example: Compound interest

Calculation and understanding of compound interest is another topic where students can benefit from programming activity. In a mathematical model, compound interest can be expressed as

$$c = p \cdot \Big((1 + r)^n - 1 \Big), \tag{1}$$

where c is the compound interest, p is the principal loan, r is the interest rate and n is the number of compounding periods. To many students, this expression resides on an isolated island of formal mathematics. Even students who can easily calculate the interest rate for one period, will not perceive compound interest as common sense.

By simulating this scenario, "walking through" each period and performing the calculations step by step, the student can gain understanding and connect the mathematical problem to the real-world problem. This simulation can be of various forms and a range of tools can be applicable including mental models and pen and paper. Spreadsheets are also well suited for this problem, however translating the acquired understanding and knowledge to the mathematical formal model is challenging. To solve this problem in a programming environment, a loop can be applied simulating each period performing the "common sense" calculations repeatedly. Text-based programming can be used (see Algorithm 1 for a Python implementation) as well as block-based environments such as Scratch [22]. A "walk" from period to period can be simulated, allowing the student to observe the effect of the compound interest.

3.3 Example: Coin toss simulator

When introducing probability, a coin toss experiment can be conducted where a (fair) coin is tossed repeatedly. Through this experiment, the student can experience that the probability of heads approaches 0.5. However, a student will often need to toss the coin many times before she is convinced that the number of heads divided by the number of tosses will converge at 0.5. With 100 coins flipped, the probability of less than 40 or more than 60 heads is 3.5 %. Hence, after 100 trials, many students will not have experienced a convergence towards the true probability. A solution, of course, is to increase the number of trials in the experiment. This is time consuming, but practically feasible through collaboration. If now it is common sense to the student that the probability of

Algorithm 1: Calculating compound interest: Simulating a walk through each year, adding interest

```
1  nof_years = 10                        # number of iterations
2  interest = 0.04                       # interest rate
3  principal = 5000                      # loan in year 0
4  current = principal                   # loan in year n
5
6  for n in range(0,nof_years):          # loops through all iter.
7      new = current*interest            # additional this year
8      current = current + new           # updates loan
9
10     print('Loan after year', n+1, 'is: ', current)
```

heads in one coin toss is 0.5, will it be straight forward to calculate the probability of two heads in two tosses? Or three heads in five tosses, given that the first one is heads? We cannot repeat the physical experiment again for each new problem. But programming can help us out. With a few simple lines of code, the student can implement this experiment and increase the number of iterations until she is convinced that the theoretical probability equals the result in the simulated experiment.

Algorithm 2: Tossing one coin

```
1  from random import randrange
2
3  nof_trials=10000                  # number of iterations
4  nof_heads=0                       # counts the number of heads
5
6  for n in range(nof_trials):       # loops through all iter.
7      coin=randrange(0,2)           # returns random 0 or 1
8      if coin==1:                   # if coin is heads
9          nof_heads+=1              # updates the counter
10
11 print(nof_heads/nof_trials)       # prints the result
```

Algorithm 2 shows how the experiment with one coin can be modelled in the programming language Python. In line 1, the `randrange` function is imported from the random package. This function allows us to pick the integers 0 and 1 randomly, and we let 1 represent heads and 0 represent tails. The program is initialized with the number of tosses specified by the user (line 3) and number

Algorithm 3: Tossing two coins

```
1  from random import randrange
2
3  nof_trials=10000                  # number of iterations
4  nof_heads=0                       # counts the number of heads
5
6  for n in range(nof_trials):       # loops through all iter.
7      coin1=randrange(0,2)          # returns random 0 or 1
8      if coin1==1:                  # if coin is heads
9          coin2=randrange(0,2)      # returns random 0 or 1
10         if coin2==1:              # if coin is heads
11             nof_heads+=1          # updates the counter
12
13 print(nof_heads/nof_trials)       # prints the result
```

of heads are set to zero (line 4). The program loops through all tosses (line 6 - 9) and in each iteration a coin is tossed, and the function `randrange` returns an integer 0 or 1 (in the range $[0, 2)$) and the number of heads are updated when the random number is equal to 1. Finally, the results are printed. By slightly changing the lines of code, we can calculate the probability of two heads in two tosses (Algorithm 3). If the first coin is heads, we toss the coin one more time (line 9), and if this is heads as well, we update the counter (line 11).

Following this algorithmic approach, it is easy for the student to build a mental version of the situation [13]. The code loops through the experiment in a way that correspond to the real-world situation, such that all steps remain common sense to the student. In contrast, explaining that the probability of two coins in two tosses equals the probability of heads in toss one multiplied by the probability of heads in toss two is more challenging.

4 Dealing with infinity and infinitesimals

Although we find physical meaning to many natural numbers, such as the number of fingers on your hand, Hersh [15] argues that if you pick a very large natural number (such as 2 to the power of a very large number raised to the power of a very large number), it is questionable what physical meaning such a number has. Hence, the natural numbers are part of an abstract theory. Yet, as long as the students can relate mathematics to the physical world, connecting the number to a physical observation, most students will experience it as "common sense". Ernest [8] argue that "If you value mathematics as we know it [...] you cannot help maintaining that infinitely many mathematical objects exist, and some of them may justly be described as infinitely large or infinitely small, infinitely far or infinitely near." But what would you answer if someone ask

where and how all these infinities are piled up? Understanding and working with the concept of infinity (a quantity that is larger than any standard real number) and infinitesimals (a nonzero quantity that is closer to zero than any standard real number) is challenging for many students.

Inspired by the history of mathematics, Gravemeijer and Doorman [12] explain how the development of calculus started with modeling everyday problems of the real-world concerning distance and velocity. These problems were initially solved using discrete approximations and discrete graphs, which later were generalized for the continuous case, forming the basis of calculus. A key challenge for many students is the translation of a discrete real-world problem to the mathematical modeling environment where the problem is solved using methods for continuous mathematics (calculus). Gravemeijer and Doorman [12] explain that the discrete functions played a key part as intermediary in the original development, and hence argue that this should be essential in the students' reinvention. However, at some point the discrete or approximate solution must be translated to the mathematical formulation of continuous mathematics, a translation which is challenging to many. Programming is a powerful tool that can aid the student in testing many more cases than what is practically feasible with manual calculations. It is easy to increase or decrease the number of iterations, and to visualize how this affects the results. Altogether, this mathematical activity can aid the student in understanding, or at least accepting, the translation from the discrete to the continuous case.

4.1 Example: Definite integrals

Let us illustrate with an example. To approximate a definite integral of $f(x)$ from a to b of a real-valued function $f(x)$, we can sum rectangles under the curve. We let the width of the rectangles approach zero, and the the number of rectangles approach infinitely, which gives the definite integral defined as the limit of a Riemann sum;

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} \left(f(x_i) \frac{b-a}{n} \right). \tag{2}$$

It is not possible to calculate the area of these rectangles, even if we continue forever. A computer cannot compute an infinite number of operations or deal with infinitesimals. It can, however, compute an arbitrary large or arbitrary small number. Hence, algorithms can find approximate solutions. We can approximate the definite integral, following the approach of summing rectangles, but now we sum finitely many. In general, we will not find the exact solution, but we can get arbitrary close by increasing the number of rectangles (n) and reducing the width of the rectangles ($b-a$).

When introducing students to the derivative or integral, tall [27] explains that many would argue that first the limit concept must be understood. But in the student's mind the introduction of the limit concept would appear for no

Algorithm 4: Numeric integration of $\int_a^b x^2 dx$ for $a = 3$ and $b = 5$ using the rectangle method

```
a=3                              # starting point
b=5                              # ending point
n=100                            # number rectangles

def f(x):                        # defining the functions
    return x**2

width=(b-a)/n                    # the width of all rectangles
A=0

for i in range(0,n):
    height= f(a+width*i)         # the height of this rectangle
    A=A+height*width             # add rectangle to total area
```

reason at this point. Alternatively, the student can first be exposed to a more qualitative, global introduction of the mathematical concepts [12]. For example, the student can first calculate the distance covered by a car traveling at constant speed, experiencing that this equals the area under the speed curve which can be found by calculating the area of a rectangle. Expanding the students' common sense, the students can gradually be exposed to more complicated speed graphs. In the most trivial cases with constant speed (and zero acceleration) setting up a physical experiment is quite easy. Many students will also be able to create a mental model [13] of such situations. However, if the acceleration is different from zero, or even dynamic, the speed can change throughout the experiment, and many students will struggle to construct accurate mental models. Here programming can be a powerful aid. The students can model and visualize the situation, and results and parameters can be updated simultaneously, allowing the student to "experience" or "witness" how it all interconnects. The students can play with the parameters of the model and see how changes in one parameter affects the results. The rectangle method used to approximate definite integral is fairly straight-forward to implement, see Algorithm 4. With this method at hand, the students can experience how changes in the speed curve affects the calculated distance between a and b and experience that this equals the area under the curve. Furthermore, they experience how the width of the rectangles ($(b-a)/n$) influences the accuracy of their estimates.

5 Algorithmic Representations

Duval [7] explains how, in other fields of knowledge such as physics, astronomy and geology, we can often gain perceptual and instrumental access to the phenomena we study without the use of semiotic representations. Formulas and

graphs can certainly be useful when we describe the motion of a car, but it is also possible to study the car's motion by driving. In mathematics, however, there is "no other ways of gaining access to the mathematical objects but to produce some semiotic representations" [7].

When teaching and studying functions, in school textbooks and other teaching materials, it is common to explore different representations [11]. Janvier [16, Ch. 3] identifies four different representations commonly used in teaching situations, that is verbal situation, table, graphs and formula. "Translation skills" required in order to move from one representation to another are also identified [23, p. 58]. Gagatsis and Shiakalli [11] argue that most researchers agree that translation ability is important for the learning of mathematics. In their study, they find that the ability to translate between the different representations of a function is associated with success in problem solving.

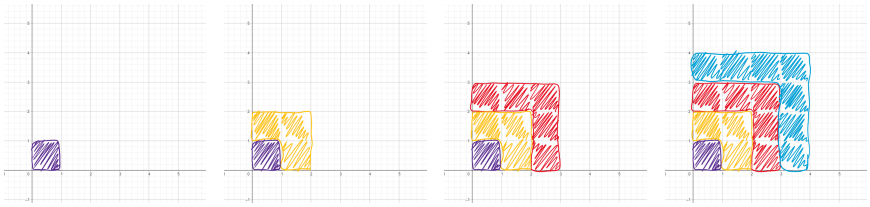
Some students fail to translate between the different representations because they lack the fundamental understanding of what a function is. Sometimes, however, the challenges with translating from one representation to another is not not due to lack of understanding, but rather due to complexity of the function. In statistics and machine learning, when regression or data-driven methods are used to identify the relationship between variables based on a dataset, the relationship can be very complicated making it impossible for humans to understand or predict the outcome [2], although various methods from the field of explainable artificial intelligence [3] are developed to help the user to gain insight into the model.

In addition to the four representations described by Janvier [16, Ch. 3] (situation, table, graph and formula) a function can be formulated in a programming language. Our belief is that in some situations this expression can be easier to identify. For example the area A of a square with side lengths $i \in \{1, 2, \dots, N\}$ can be calculated by adding $2i - 1$ to the previously calculated area. For example, the area A of a square with side length $N \in \mathbb{N}$ can be calculated by adding $2N-1$ to the area of the square with side lengths $N-1$. From this, we can obtain an incremental algorithm for calculating the area, as shown in Algorithm 5 and Figure 3.

Algorithm 5: Area of a square for $i \in \{1, 2, \dots, 100\}$

```
A=0
for i in range(1,100):
    A=A+i*2-1
    print(A)
```

Although it is considered useful to work with different representations, students are of course not faced with all representations in every situation. Some-



(a) $i = 1$ (b) $i = 2$ (c) $i = 3$ (d) $i = 4$

Fig. 3: Area of a square with side lengths i. For each increment of i, the area increases by $2i - 1$.

times, for example at an introductory level, focusing on a table and situation, and the translation between these two representations, can be suitable.

6 Discussion

If the goal of introducing and teaching programming is increased performance in mathematical relational understanding [26] and ability to solve problems in a traditional mathematics curriculum, is it then worthwhile? Will programming benefit student achievement in mathematics more than traditional mathematics education? For future research we intend to explore this by conducting comparative experiments. This is challenging, and although previous studies have shown positive effects on mathematics performance, the results are difficult to generalize [9]. As programming and mathematics are tightly interconnected, activity in one domain will often influence performance in the other. Hence, it is not enough to measure a positive correlation between programming activities and mathematical knowledge and skills. We also have to consider to what degree the programming activity and teaching contribute to relational understanding, and compare the effectiveness of programming as a strategy to increase mathematical performance.

Several studies indicate that programming and computer games activities lead to positive effects on the students' attitudes, including self-confidence, value, enjoyment, and motivation, towards mathematics [20]. Based on a review of 15 selected articles, Forsström and Kaufmann [9] argue that some studies found that programming improved the students' motivation. It was however not possible to generalize these results. This was partly because the programming activities reported in the studies were different from regular classrooms activities. Furthermore, they did not report how motivation developed when the programming activity was integrated into normal classrooms routines.

Lindh and Holgersson [21] studied the effect of a one-year regular robotic toys training and found no obvious over-all effect on the pupils' ability to solve

mathematical and logical problems, although significant positive effects for sub groups of pupils were reported. Different programming activities will affect the students' mathematics performance differently. How programming is introduced and taught is also of importance.

7 Conclusions

In the beginning of this paper, we discuss challenges related to the gap between formal mathematics and the learner's real-world experiences. We also argue that mathematical activity and guided reinvention can facilitate new knowledge and skills. Nevertheless, teachers will experience that when guiding students in their reinvention, many students will fail to reinvent formal mathematics, and formal mathematics can be perceived as an island which is separated from the mainland of real-world experiences.

This paper proposes that programming can be utilized in bridging this gap. This is illustrated by a set of classical mathematics problems which we solve using algorithms consisting of non-complex operations which are performed repeatedly. For example when teaching probability, the students can toss a set of coins repeatedly to estimate its probability distribution (equal probability for heads and tails). But the number of times a student can toss a set of coins is limited. Computers, however, don't get bored or tired, and we can utilize this to perform simple tasks or instructions repeatedly. Similarly, students can experience and observe how the sum of rectangles under a curve will converge to the area under the curve as the number of rectangles is increased (and the width of each rectangle is decreased). Integrals are perceived as abstract and complex, but the rectangle method is only a collection of non-complex operations which are performed repeatedly.

Through programming, the students can experience and observe how the results develop when the number of iterations increases. Our proposal is that this repetition can serve as a powerful didactic technique which can facilitate understanding and comprehension of formal mathematical concepts.

Acknowledgements

Criticisms and discussions regarding the content and topics of this paper with Runar Berge (Volda University College) and Arne Kåre Topphol (Volda University College) are greatly appreciated.

References

1. Berland, M., Wilensky, U.: Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. Journal of Science Education and Technology **24**(5), 628–647 (2015)

2. Brandsæter, A., Knutsen, K.E.: Towards a framework for assurance of autonomous navigation systems in the maritime industry. In: Safety and Reliability–Safe Societies in a Changing World : Proceedings of ESREL 2018, pp. 449–457. CRC Press (2018)
3. Brandsæter, A., Glad, I.K.: Explainable artificial intelligence: How subsets of the training data affect a prediction. arXiv:2012.03625 (2020)
4. Brandsæter, A., Osen, O.L.: Assessing autonomous ship navigation using bridge simulators enhanced by cycle-consistent adversarial networks. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability **0**(0), 1748006X211021040 (2021). https://doi.org/10.1177/1748006X211021040, https://doi.org/10.1177/1748006X211021040
5. Dall'Alba, G., Walsh, E., Bowden, J., Martin, E., Masters, G., Ramsden, P., Stephanou, A.: Textbook treatments and students' understanding of acceleration. Journal of research in science teaching **30**(7), 621–635 (1993)
6. Doorman, M.: How to guide students? A reinvention course on modeling motion. In: Common Sense in Mathematics Education. Proceedings of 2001 The Netherlands and Taiwan Conference on Mathematics Education. pp. 97–114 (2002)
7. Duval, R.: Representation, vision and visualization: Cognitive functions in mathematical thinking. basic issues for learning. In: Proceedings of the Annual Meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education (21st, Cuernavaca, Morelos, Mexico, October 23-26, 1999) (1999)
8. Ernest, P.: Mathematics education and philosophy: An international perspective. Routledge (2003)
9. Forsström, S.E., Kaufmann, O.T.: A literature review exploring the use of programming in mathematics education (2018)
10. Freudenthal, H.: Revisiting Mathematics Education: China Lectures Mathematics Education Library ; V. 9. Springer, Dordrecht (1991)
11. Gagatsis, A., Shiakalli, M.: Ability to translate from one representation of the concept of function to another and mathematical problem solving. Educational psychology **24**(5), 645–657 (2004)
12. Gravemeijer, K., Doorman, M.: Context problems in realistic mathematics education: A calculus course as an example. Educational studies in mathematics **39**(1), 111–129 (1999)
13. Greeno, J.G.: Number sense as situated knowing in a conceptual domain. Journal for research in mathematics education **22**(3), 170–218 (1991)
14. Grover, S., Pea, R.: Computational thinking in k–12: A review of the state of the field. Educational researcher **42**(1), 38–43 (2013)
15. Hersh, R.: What is mathematics, really? Mitteilungen der Deutschen Mathematiker-Vereinigung **6**(2), 13–14 (1998). https://doi.org/doi:10.1515/dmvm-1998-0205, https://doi.org/10.1515/dmvm-1998-0205
16. Janvier, C.: The interpretation of complex cartesian graphs representing situations: studies and teaching experiments. Ph.D. thesis, Nottingham University (1978)
17. Kahneman, D.: Thinking, fast and slow (2011)
18. Kaput, J.J.: The representational roles of technology in connecting mathematics with authentic experience. Didactics of mathematics as a scientific discipline pp. 379–397 (1994)
19. Kaufmann, O.T., Stenseth, B.: Programming in mathematics education. International Journal of Mathematical Education in Science and Technology pp. 1–20 (2020)

20. Ke, F.: An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. Computers & education **73**, 26–39 (2014)
21. Lindh, J., Holgersson, T.: Does lego training stimulate pupils' ability to solve logical problems? Computers & education **49**(4), 1097–1111 (2007)
22. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The scratch programming language and environment. ACM Transactions on Computing Education (TOCE) **10**(4), 1–15 (2010)
23. Nilsen, H.K.: Learning and teaching functions and the transition from lower secondary to upper secondary school. Ph.D. thesis, University of Agder (2013), https://uia.brage.unit.no/uia-xmlui/handle/11250/139758
24. Norstein, A.: Bruk av excel og geogebra til utforsking i matematikkfaget. In: Norstein, A., Haara, F.O. (eds.) Matematikkundervisning i en digital verden, chap. 3, pp. 51–72. Cappelen Damm Akademisk (2018)
25. Sandene, I.M., Haara, F.: Likningsløysing med dragonbox. In: Norstein, A., Haara, F.O. (eds.) Matematikkundervisning i en digital verden, chap. 2, pp. 27–50. Cappelen Damm Akademisk (2018)
26. Skemp, R.R.: Relational understanding and instrumental understanding. Mathematics teaching in the middle school **12**(2), 88–95 (2006)
27. Tall, D.: Advanced mathematical thinking, vol. 11. Springer Science & Business Media (1991)
28. Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., Wilensky, U.: Defining computational thinking for mathematics and science classrooms. Journal of science education and technology **25**(1), 127–147 (2016)
29. Wubbels, T., Korthagen, F., Broekman, H.: Preparing teachers for realistic mathematics education. Educational studies in mathematics **32**(1), 1–28 (1997)