Training teachers in Informatics: a central problem in science education

Sylvia da Rosa, Marcos Viera, and Juan García-Garland

Instituto de Computación, Facultad de Ingeniería, Universidad de la República.
{darosa,mviera,jpgarcia}@fing.edu.uy

Abstract. In this paper we describe a didactic approach that seeks to include the stages of programming in the process of problem solving in mathematics and physics. Secondary education teachers were introduced to the strategy in a teacher training course that emphasises the importance of discrete mathematics and logic as the foundations of Computer Science concepts. A functional programming language -named MateFun- was used to explore the goals of the didactic strategy. The language can animate figures, a feature useful for physics learning experiences in the sense that visual representations of solutions to problems can be programmed. The didactic approach introduces programming into the learning experience beyond the instrumental use of technology and contributes to a natural introduction of Computer Science in other subjects. The paper contains examples and conclusions.

1 Introduction

The introduction of Computer Science in education is generally conceived as a technological issue. As a result, we find educational approaches that place Computer Science at the instrumental level; as a tool to be used to support learning in other subjects through the application of Computer Science. This approach to integration can result in very successful learning experiences, provided special care is given to planning and execution.

However, integration at the instrumental level is a limited approach that fails to introduce students to the foundations of Computer Science, including logic, discrete mathematics and the history and philosophy of Computer Science. According to our experience, first year University students have serious difficulties to understand concepts related to logic and discrete mathematics. For instance, they are restricted in their understanding of function to the idea of an algebraic formula for computing values; they hardly ever use either quantifiers carefully in proofs or notions of logic in calculations (i.e. boolean functions); and they treat induction as a recipe to solve certain problems, often involving sums of numbers.

The digital era our students face today requires certain competencies which can only be delivered through a more holistic integration of Computer Science in education.

Informatics Europe[1] analyzed several reports from 2013 to 2020 about Informatics education in Europeans countries concluding that despite advances much needs to be done before a real solution can be achieved [10].

The work cited above includes a position paper from 2020 [14] where the authors point out that "... teaching Informatics to all, both as independent subject and integrated in other subjects, calls for a need to rethink in overall terms what to teach (both breadth and depth) and how to teach it." They address the point through mainly three challenges that we summarise as follows:

- Define schools curricula in a way that respects the process of learning,
- Educate and support teachers at all levels to teach a discipline,
- Test and verify that both previous items are appropriately designed for the various levels of education. This aspect requires didactic investigations.

The education system in our country faces these challenges as well.

In this paper we describe a programming course aimed at teachers of mathematics and physics that tries to address some of the problems mentioned in this introduction by proposing a didactic strategy that explores:

- how to formulate algorithmic problems
- how to design solutions (algorithms) and make efficient programs
- how to interpret physical processes as information processes that can be simulated on a computer

In other words, the goal is for teachers to learn the basis of Computer Science and experience the benefits of such learning. Learning Computer Science means learning how to create new technologies, rather than simply using them, and how to do computational science [5, 12].

The programming language used in the course is part of the functional paradigm, in the understanding that this is a good vehicle to introduce Computer Science concepts related to their mathematical foundations. Moreover, mathematics and physics courses in secondary education are sources of several types of algorithmic problems whose solutions can be easily programmed and visualised.

The sections of the paper are organised as follows: In the next section we present a brief description of the main features of MateFun. In Section 3 we introduce the main characteristics of the course and the didactic strategy, while in Section 4 we present some interesting math and physics exercises done by the teachers. Finally, we conclude with considerations for future work.

2 MateFun

Since teachers have normally no knowledge of programming we use the language MateFun [3], which is a programming language designed with the aim to be a tool to learn mathematical functions. Its syntax is very similar to mathematics, and its use encourages the goals of the activities.

[1] Informatics Europa is part of The Informatics for All Coalition, with ACM Europe Council and Council of European Professional Informatics Societies (CEPIS).

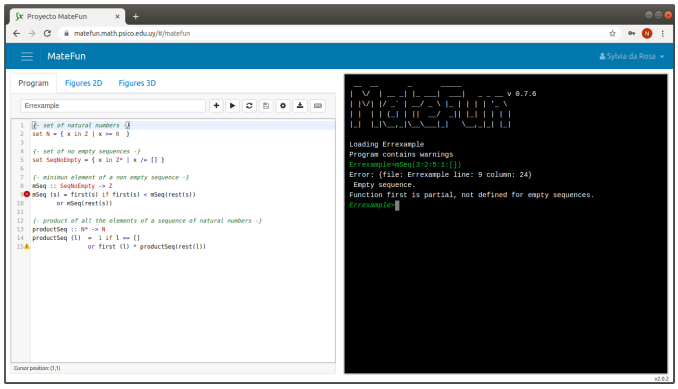

Fig. 1: MateFun IDE

2.1 Description

MateFun is purely functional, meaning that functions do not introduce side effects and they only depend on their arguments. MateFun is an interpreted language. To be easily approachable it is available as a web integrated development environment (Matefun IDE[2]), as shown in Figure 1. The left framework is where the program is written and the right framework is where the expressions are evaluated.

Syntax and semantics of MateFun are both influenced by the seek to be a tool to express mathematics. The syntax is minimal and near to the usual mathematical notation. Semantically, it has the peculiarity of being strongly typed, while having no type inference. The skill to specify the domain and range of a function is part of the learning process when learning about functions. In MateFun type information must be given by users.

A MateFun script is a list of set definitions and function definitions over such sets. Predefined sets such as `R` (representing real numbers) or `Z` (representing integer numbers) are available as built-in constructs.

The user can define new sets either by comprehension or by extension just as usually presented in mathematics courses. In the following example we define the sets of natural numbers (`N`), non-zero real numbers (`Rno0`) and days of the week (`Day`).

```
1 set N    = { x in Z | x >= 0 }
2 set Rno0 = { x in R | x /= 0 }
3 set Day  = { Mon, Tue, Wed, Thu, Fri, Sat, Sun }
```

Sets such as `Rno0`, defined by comprehension, take a base set (`R` in this case) and refine them with a predicate. Predicates can be composed using relational operators and conjunctions of predicates.

Functions are defined giving a signature and a proper definition. For instance, one could define the inverse function over the -nonzero- real numbers:

[2] `https://fing.edu.uy/proyectos/matefun/#/en/login`

```
f :: Rno0 -> R
f (x) = 1/x
```

MateFun supports some of the usual idioms used to define functions in mathematics. For instance, piecewise functions can be defined. The following MateFun definition specifies the absolute value function over the real numbers:

```
abs :: R -> R
abs (x) =   x if x >= 0
          or -x
```

This program resembles the definition in the usual mathematical notation given by:

$$abs : \mathbb{R} \to \mathbb{R}$$
$$abs(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$$

Piecewise function definitions are the standard way to define functions in high school mathematics. For this reason, MateFun supports neither pattern matching nor conditional expressions, avoiding extra constructs in the language.

To emphasise that not all functions are numeric, MateFun allows to define non-numeric sets (eg. `Day`) and functions between those sets, such as `nextDay`:

```
nextDay :: Day -> Day
nextDay (d) =  Tue if d == Mon
            or Wed if d == Tue
            or Thu if d == Wed
            or Fri if d == Thu
            or Sat if d == Fri
            or Sun if d == Sat
            or Mon
```

Functions with multiple variables can be defined using *n-tuples* (the generalization of cartesian products). For example, the following function computes the area of a triangle, given its base and height:

```
areaTria :: R X R -> R
areaTria (b, altura)  = (base * altura) / 2
```

We can also define *sequences* of elements of a given set; usually called *lists* in programming. For instance, the sequence set `N*` (sequence of naturals) is defined inductively as:

- `[]`, the empty sequence
- `n:ns`, a sequence composed by an element `n` belonging to `N` and a sequence `ns` belonging to `N*`.

There exist some primitive functions to operate with sequences: `first(s)` returns the first element of the sequence `s`, `rest(s)` returns the sequence `s` without its first element, and `range(n,m,k)` returns a sequence of numbers $(n, n+k, n+$

$2k, ...$) from `n` to `m` with step `k`. With `range`, combined with a function to `sum` the elements of a sequence, we can for instance easily implement the summatory $\sum_{i=m}^{n} i$.

```
summatory :: N X N -> N
summatory (m, n) = sum(range(m, n, 1))

sum :: R* -> R
sum (xs) =  0 if xs == []
          or first(xs) + sum(rest(xs))
```

Notice the use of recursion in the implementation of `sum`.

The language includes the sets `Figure` and `Color`, and a set of primitive functions to create and transform figures. For example, the following function returns a red-colored circle of a given radius, centered in the $(0, 0)$ point of a Cartesian plane.

```
redCirc :: R -> Fig
redCirc (r) = color(circ(r), Red)
```

Animations are sequences of figures. The following function takes a figure and a number `n` of steps, and returns an animation in which the figure is moved `n` times one step to the right in the x-axis:

```
moveRight :: Fig X Z -> Fig*
moveRight (f, n)  =  [] if n == 0
                     or f : moveRight(move(f,(1,0)),n-1)
```

3 Main features of the course

The course is developed using the Moodle platform, during three months in which some video conferences instances[3] take place. The participants work in groups and they are assisted through the forums. In the first part of the course (approximately 6 weeks) they solve exercises to practice and learn the language MateFun. To assess the degree of learning achieved with the exercises, teachers must submit an assignment. In the rest of the course, the teachers have to design a didactic sequence of activities and carry them out with their students. These activities are evaluated as final work.

One of the main characteristics of the course is that teachers' knowledge and opinions are taken into account, in the sense that course teachers (university researchers) provide tools and general guidelines, and the participants design the didactic instances of the final work, as well as the modalities of work with their students. In this way, an attempt is made to promote a collaborative learning community where knowledge is co-produced, rather than imposing ideas.

[3] Because of covid 19, normally these are face to face instances.

The general guidelines seek to facilitate the learning of mathematical concepts that are the foundation of Computer Science. We understand that programming a solution to a problem reveals aspects of the resolution process that are otherwise unconscious. At the same time, the fact that the problem is firstly mathematically solved, encourages the acquisition of good programming practices and discourages the idea that to program is "just making programs work".

3.1 The didactic strategy

In order to specify the approach used to build the teacher training course, we will outline the main features of the didactic strategy: First; we ground the approach on the definition of algorithmic problem as written by [8]:

An algorithmic problem consists of:

1. a characterisation of a legal, possibly infinite, collection of potential input sets, and
2. a specification of the desired outputs as a function of the inputs

The function by which the input data is converted into the required result is the algorithm that solves the problem. In other words, an algorithm is a solution to an algorithmic problem.

Hence, the problem-solving process begins with the interpretation of the statement in terms of input-output as a first approximation towards thinking computationally.

Programming in MateFun requires teachers to rigorously state the problem, including the signature of functions (input and output). This is particularly important when considering how secondary education teachers will often state problems forgetting details that are taken for granted (for instance domain and co-domain of functions). Indeed, using a programming language will develop skills in rigorous problem formulation through practices such as handling the computer, finding symbols on the keyboard, the syntax, etc.

Second; we pay special attention to the design of solutions (algorithms) and efficient programming. The starting point here is the teachers' own solutions to the stated problem. From these we introduce basic notions of algorithmic complexity and program correctness, and discuss how more efficient solutions can be obtained. We illustrate this point in the next section with the definition of `firstM2` function, introduced from `firstM` as given by the teacher. Once the teacher has arrived at multiple solutions, we introduce the principle of structural induction and demonstrate how it can be used to prove that the definitions are equivalent, as shown in 4. More examples can be found in [16].

The third and final feature of our didactic strategy is its broad applicability. While we initially sought to produce a didactic practice which brings programming into Mathematics lessons, we have gradually developed an approach which can be used in other disciplines. In the examples we present in the next section, we includes a physics exercise which illustrates this point 4.3. Furthermore, we can see the benefits of introducing similar approaches in other sciences as well.

Following the vision of P. Denning and M. Tedre and their review of how *computational thinking became central to the sciences*, we seek to bring the experiences and practices of computational science to science education and teaching more broadly [5].

4 Selected exercises

In this section we present some examples of exercises that mathematics and physics teachers did with their students in the classroom. The posed problems are mostly taken from their courses so they know how to solve them and they learn how to implement the algorithms in MateFun. The topics that teachers choose vary according to the level of their groups (from 1st. to 6th. of secondary education). They have to take also account of the program of the school year. Popular topics are divisibility, lineal and quadratic equations, statistic, successions, analytic geometry, among others. Physics teachers take advantage of MateFun facilities to create graphics, figures and animations to illustrate concepts of their subject.

4.1 Successions

One of the topics where the mathematics teachers found that programming helps students' understanding is related with successions. As example we include the following exercise:
Given the succession below:

$$a_n : \mathbb{N} \to \mathbb{R}$$
$$a_n = \begin{cases} 1 & \text{if } n = 1 \\ ((n-1)/n) * a_{n-1} & \text{otherwise} \end{cases}$$

1. write a MateFun function to obtain any term.
2. write a MateFun function that given a value `m`, returns a sequence with the first `m` terms of the succession.
3. write a MateFun function that given a value `m` calculates the sum of the first `m` terms of the succession.

Below are some of the solutions that teachers worked with the students in the classroom.

```
30 {-Part 1: any term of the succession -}
31 anyTerm :: N -> R
32 anyTerm (n) =  1 if n == 1
33              or ((n-1) / n) * anyTerm(n-1)
34
35 {- Part 2: sequence of the first m terms of the
   succession -}
36 firstM :: N -> R*
```

```
firstM (m) =  []  if m == 0
           or  anyTerm(m) : firstM(m-1)

{- Part 3: sum of the first m terms of the succession.
-}
sumFirstM :: N -> R
sumFirstM (m) = sum(firstM(m))
```

For part 2 we introduce `firstM2`, a solution that merges the construction of the sequence with the formula to construct a term. Although more efficient, this solution sacrifices clarity in the definition with respect to `firstM`.

```

{- A more efficient definition of the function of part 2
 -}
firstM2 :: N -> R*
firstM2 (m) =  []  if m == 0
            or  addTerm(m,firstM2(m-1))

addTerm :: N X R* -> R*
addTerm (n,rs) =  1:rs if n == 1
               or ((n-1) / n) * first(rs) : rs
```

This is an interesting new computational dilemma for teachers, that illustrates the second point of our strategy.

This kind of exercises help the students in constructing the concept of succession as a function from $\mathbb{N}$ to any set; $\mathbb{R}$ in the case of the example. The third part of the exercise is also solved with the formula below, which is traditionally presented to students.

$$\sum_{i=1}^{m} a_i = m * (a_1 + a_m)/2$$

Students are encouraged to implement it in MateFun and with teacher's help they arrive to:

```
sumFirstM' :: N -> R
sumFirstM' (m) = m * (anyTerm(1) + anyTerm(m)) / 2
```

Since both definitions (`sumFirstM` and `sumFirstM'`) reveal two methods of calculating the sum of terms in a succession, in our course teachers are asked to prove that these are equivalent. The following property is stated:

Property 1. $\forall$ `n` $\in$ `N`, `sumFirstM(n) = sumFirstM'(n)`.

The property is proved using the principle of structural induction:

1. Base case: prove property for `n` $= 1$
2. Inductive case: $\forall$ `n` > 1, if `sumFirstM(n) = sumFirstM'(n)` then `sumFirstM(n+1) = sumFirstM'(n+1)`
3. If 1) and 2) then Property holds.

Base case:

$$
\begin{aligned}
& \texttt{sumFirstM}(1) \\
= \; & \texttt{sum}(\texttt{FirstM}(1)) && \{ \text{ def. } \texttt{sumFirstM} \} \\
= \; & \texttt{sum}(\texttt{anyTerm}(1) : []) && \{ \text{ def. } \texttt{FirstM} \} \\
= \; & \texttt{sum}(1 : []) && \{ \text{ def. } \texttt{anyTerm} \} \\
= \; & \texttt{first}(1 : []) + \texttt{sum}(\texttt{rest}(1 : [])) && \{ \text{ def. } \texttt{sum} \} \\
= \; & 1 + \texttt{sum}([]) && \{ \text{ def. } (\texttt{first}, \texttt{rest}) \} \\
= \; & 1 + 0 && \{ \text{ def. } \texttt{sum} \} \\
= \; & 1 * (1 + 1)/2 && \{ \text{ arith } \} \\
= \; & 1 * (\texttt{anyTerm}(1) + \texttt{anyTerm}(1))/2 && \{ \text{ def. } \texttt{anyTerm} \} \\
= \; & \texttt{sumFirstM}'(1) && \{ \text{ def. } \texttt{anyTerm'} \}
\end{aligned}
$$

For space reasons the proof of the inductive case is not included[4]. Observe that in the language used to prove such properties every step has to be well founded, stating a rigorous equational reasoning. Through these exercises, the relationship between mathematics and Computer Science comes out in an easily understandable way. Even more, teachers could introduce their students in the topic of inductive proof in cases they consider adequate.

4.2 Advanced examples

The teachers usually work with GeoGebra [9] that is an interactive geometry, algebra, statistics and calculus application, intended for learning and teaching mathematics and science from primary school to university level. They find a challenge in solving exercises that they have previously solved with GeoGebra. It is worth saying that some teachers give a great importance to the possibility of solving exercises and drawing the solution as they do with GeoGebra. We point out that a programming language like MateFun allows to construct the mathematical solution as an object - *the program* - putting into practice from more basic notions to advanced concepts. Even more, the program can be executed and we can see our solutions in action. The example below illustrates the case.

Given a function f it is possible to compute an approximation of the definite integral

$$\int_a^b f(x)dx$$

using the *rectangle method* (or *Riemann sum*). The method consists of partitioning the interval $[a, b]$ in n equidistant sub-intervals $[x_i, x_{i+1}]$ where $i \in \{0 \dots n\}$ $x_i = a + i\left(\frac{b-a}{n}\right)$. For each sub-interval a point x^* is chosen. A rectangle of width $\frac{b-a}{n}$ and height $f(x^*)$ is an aproximation of $\int_{x_i}^{x_{i+1}} f(x)dx$ and the sum of rectangles approximates the full integral, improving with a bigger n.

[4] It remains as an exercise for the reader.

Taking the leftmost point at each interval (left Riemann sum) the approximation is given by the expression:

$$\sum_{i=1}^{n} f(a + (i-1) * w) * w$$

where $w = \frac{b-a}{n}$

Given, for example, the function $f(x) = \frac{x^2}{10}$ the problem is solved in MateFun in the following way.

```
f :: R -> R
f (x) = 0.1 * x ^ 2

integral_f :: R X R X N -> R
integral_f (a, b, n) = summatory_f(1, n, a, (b - a) / n)
```

where `summatory_f` is a function computing recursively the sum of the areas of the $n - i + 1$ rightmost rectangles, implemented as:

```
summatory_f :: N X N X R X R -> R
summatory_f (i, n, a, w)
  =  0 if i > n
  or f(a + (i-1) * w) * w + summatory_f(i+1, n, a, w)
```

Another version, with a more compositional style, using the function `sum` can be implemented. First, compute the sequence of areas:

```
areas :: R X R X R -> R*
areas (a,n,w)  =  [] if n == 0
               or f(a) * w : areas(a+w, n-1, w)
```

And then apply the known function `sum`.

```
integral :: R X R X N -> R
integral(a,b,n) = sum(areas(a, n, (b-a)/n))
```

It is possible to plot a representation of the rectangles used to approximate the area. The result for twenty rectangles in the interval 0-10 is showed in Figure 2.

This is accomplished by the following functions:

```

drawArea :: R X R X R -> Fig
drawArea (a, b, n) = joinFigs(rectangles(a, n, (b-a)/n))

rectangles :: R X R X R -> Fig*
rectangles (a, n, w)
 =  [] if n == 0
 or rectBase(a,w,abs(f(a+w/2))) : rectangles(a+w,n-1,w)

rectBase :: R X R X R -> Fig
rectBase (i, b, h) = move(rect(b, h), (i, h/2))
```

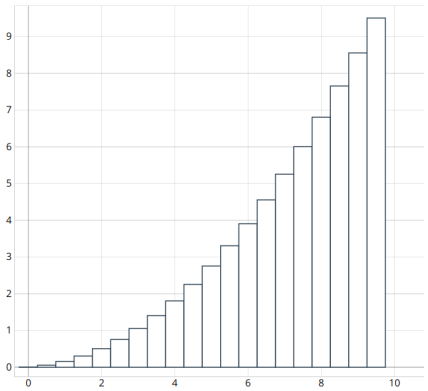

Fig. 2: `drawArea(0,10,20)`

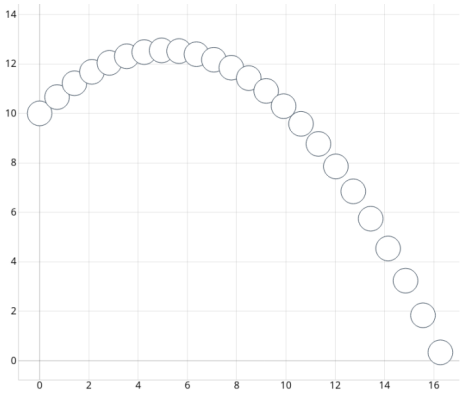

Fig. 3: `projectile(45,10,0,10,0.1))`

Through this example the power of MateFun can be appreciated with clarity: not only it makes possible to implement the solution of the problem using a syntax similar to mathematics, but also the visual representation of the solution can be programmed by the student. In other words, MateFun allows to program the graphic representations in contrast to other tools in which these are provided to the user.

4.3 Two dimensional kinematics

As an example of a physics application we present how an animation of two-dimensional projectile kinematics can be implemented.

In MateFun, animations are sequences of figures. When a value of type `Fig*` is evaluated the animation is displayed to the user. The goal in this particular exercise is to compute a sequence of figures showing the position of the projectile for uniformly increasing values of time t.

Let us consider an object thrown from the position $(0, y_0)$ with an initial speed of magnitude v_0 with an angle α from to the x axis; i.e. the initial speed vector is (v_o, α) in polar coordinates, or $(v_0 \cos\alpha, v_0 \sin\alpha)$ in euclidean coordinates. The unit of measure for distance is the meter, for angles is the degree.

To compute the position of the projectile the motion is decomposed among the horizontal and vertical components. In the horizontal component the speed is constant. Given the parameters α and v_0 and a time t relative from the launch time, the position can be computed by the following formula:

```
1 posX :: R X RPos X RPos -> R
2 posX (alpha,v0,t) = v0 * cos(alpha) * t
```

Vertically, the motion is constantly accelerated thanks to the gravitational acceleration g. Currently, users can define functions and sets in the MateFun Language, but not constants. To define a constant the unit set "`()`" -a set with an unique member denoted by "`()`"- comes handy:

```
g :: () -> R
g () = 9.8
```

Given the parameters α, v_0, y_0 and t we can compute the vertical position relative to the initial coordinate:

```
posY :: R X RPos X RPos X RPos -> R
posY (alpha, v0, y0, t)
  = y0 + v0 * sin(alpha) * t - g() * t^2 / 2
```

Finally the sample of images representing the projectile motion can be computed. Given the initial angle α, an initial speed v_0, a start time t (usually 0), the height y_0 of where the projectile is launched, and a positive number dt indicating the period (the time between frames); the `projectile` function computes the animation:

```
projectile :: R X RPos X RPos X RPos X RPos -> Fig*
projectile (alpha, v0, t, y0, dt)
   =  [] if y < 0
   or move(circ(0.5),(x,y))
       : projectile(alpha, v0, t + dt, y0, dt)
   where x = posX(alpha, v0, t)
         y = posY(alpha, v0, y0, t)
```

The object is represented as a circle. The animation ends when the object reaches the ground (given by the line $y = 0$). In Figure 3 we present a representation of the result of calling `projectile(45,10,0,10,0.1))`; while we cannot embed animations in this document, since an animation is a list of figures, we used the function `joinFigs` to generate a "trail" with the full movement of the projectile. We presented the example in two dimensions for simplicity, but note that this exercise can be extended to a three-dimensional space similarly.

From the didactic point of view, we can see in Figure 3 that at the beginning of the fall the circles look very close and as the projectile falls, at the same time intervals, their positions are increasingly separated. This clearly illustrates the effect of acceleration on the trajectory of the projectile. Through examples like this, teachers can develop two of the main competencies to do computational science, that is, modelling and simulating natural phenomena.

5 Conclusions and further work

The conclusions that can be drawn go in two directions, on the one hand, the course is an adequate approximation to provide students with basic programming knowledge. Although Computer Science should be included in secondary education as an academic discipline taught by Computer Science teachers, the difficulties still presented lead us to reaffirm what has been argued elsewhere about the role of science and mathematics teachers might have in this task [2, 4, 6, 7, 13].

On the other hand, our approach agrees with the ideas of Peter Denning and Matti Tedre set forth in [5]. Especially in Chapter 7, where the authors

describe the close relationship between Computer Science, natural science and engineering, they write:

> Computation has proved so productive for advancement of science and for engineering that virtually every field of science and engineering has developed a "computational" branch.

Our contribution is to begin to build that relationship from the early stages of education.

Finally, it is worth saying that from a didactic point of view MateFun has shown to be a really helpful tool for understanding the process of solving problems, without adding technical complications as is often pointed out about other tools. Features like a type system providing a way of defining functions including the signature (a lack of many languages, for example python!); the error messages (usually a difficulty to teachers and students) and the possibility of programming visual representations of formulas (through graphics and animations), make it easier for teachers to introduce the stages of programming and motivates the student. This is a twofold benefit: teachers recognise the impact of programming in learning science and mathematics, and at the same time basic programming knowledge becomes accessible to a great number of students.

Teaching functional programming and mathematics together is not a new idea, on the contrary, several authors since many years ago have used diverse languages to implement their purposes, notably [11, 17]. To strengthening the link between mathematics and computer science both in university level studies as in secondary schools has been also a concern of computing teachers [1,15]. One of our goals is to create an interdisciplinary community of computing researchers, science teachers and students of STEM disciplines (Science, Technology, Engineering and Mathematics). Developing tools according to educational practices of our teachers and students -as MateFun- is a factor that stimulates the interchange of ideas and facilitates the mutual comprehension of viewpoints and difficulties.

The future work is mainly oriented along two lines. First, MateFun is still in development and the activities with teachers and students provide key contributions to detect weaknesses and introduce improvements. Second, the activities carried out have to be compiled to obtain sufficient material and experience. We aspire to extend the approach to include teachers of other sciences, such as chemistry and biology, in order to strengthen the integration of science, mathematics and programming as part of teacher training.

Acknowledgment. We acknowledge Manuela Cabezas for assistance in editing the manuscript.

References

1. Baldwin, D., Walker, H.M., Henderson, P.B.: The Role of Mathematics in Computer Science. ACM Inroads Volume 4 Issue 4 pp. 74–80 (2013)

2. Bradshaw, P., Woollard, J.: Computing at School: An Emergent Community of Practice for a Re-Emergent Subject. In: International Conference on ICT in Education (2012)
3. Carboni, A., Koleszar, V., Tejera, G., Viera, M., Wagner, J.: Matefun: Functional programming and math with adolescents. In: Conferencia Latinoamericana de Informática (CLEI 2018) - SIESC (2018)
4. CSTA K12 Computer Science Standards. http://www.csteachers.org/?page=CSTA_Standards (2011)
5. Denning, P., Tedre, M.: Computational Thinking. Cambridge, MA : The MIT Press (2019)
6. Dowek, G.: Quelle informatique enseigner au lycée? Bulletin de l'APMEP, nr. 480 (2005)
7. Dowek, G.: L'enseignement de l'informatique en France, Il est urgent de ne plus attendre. https://www.academie-sciences.fr/pdf/rapport/rads_0513.pdf (2013), rapport de l'Académie des Sciences
8. Harel, D., Feldman, Y.: AlgorithmicsThe Spirit of Computing. Addison-Wesley. An imprint of Pearson Education Limited (2004)
9. Hohenwarter, M., Jones, K.: Ways of linking geometry and algebra, the case of geogebra. Proceedings of the British Society for Research into Learning Mathematics **27**(3), 126–131 (November 2007)
10. Informatics Europe, Informatics Education in Europe: Are We All In The Same Boat? https://www.informatics-europe.org/component/phocadownload/category/10-reports.html?download=60:cece-report
11. O'Donnell, J., Hall, C., Page, R.: Discrete Mathematics Using a Computer, Second Edition. Springer-Verlag London Limited 2006, ISBN-10: 1-84628-241-1 (2006)
12. Papert, S.: Mindstorms - children, computers and powerful ideas. Basic Books, Inc., Publishers / New York (1980)
13. Peyton Jones, S.: Bringing Computer Science Back into Schools: Lessons from the UK. SIGCSE'13 (2013)
14. Informatics for All: Educating People for the Digital Age. https://www.informaticsforall.org/wp-content/uploads/2020/07/Informatics-for-All-position-paper.pdf
15. da Rosa, S.: Designing Algorithms in High School Mathematics. Lecture Notes in Computer Science, vol. 3294, Springer-Verlag (2004)
16. da Rosa, S., Viera, M., García-Garland, J.: A case of teaching practice founded on a theoretical model. Lecture Notes in Computer Science 12518 from proceedings of the International Conference on Informatics in School: Situation, Evaluation, Problems pp. 146–157 (2020)
17. VanDrunen, T.: Functional programming as a discrete mathematics topic. ACM Inroads Volume 8 Issue 2 (2017)