# Functional programming and didactics of computational sciences

Sylvia da Rosa, Manuela Cabezas, Marcos Viera, and Federico Gómez

[1] Instituto de Computación, Facultad de Ingeniería, Universidad de la República.
{darosa, mviera, fgfrois}@fing.edu.uy
[2] Facultad de Educación, Universidad de la Empresa. {mcabezas}@ude.edu.uy

**Abstract.** The benefits of using multiple representation forms as a strategy to teach domains-specific topics and support a deeper, more abstract understanding of difficult content are well known in mathematics and science education. In this paper, we argue that additional benefits can be derived from multimodel strategies, where the didactic focus is placed on the relationship and interaction between mathematical and computational models, and that these are necessary for the discipline-specific knowledge domains of science education in the computational age. We propose a didactic model for teaching science (Physics, Chemistry, Biology) from knowledge and strategies applied in the mathematical and computational modelling process of real (Physics, Chemistry, Biology) phenomena/problems. We describe the development of the model through an example of Physics, and present one more example of applying the didactic model in Chemistry. For implementing the computational model, the pure functional programming language MateFun was used. The language was designed by researchers at the Institute of Computing (InCo) of the Faculty of Engineering (FING) UDELAR and is aimed at learning mathematical functions.

**Keywords:** Computational Model · Didactic Sequences · Functional Programming.

## 1 Introduction

The benefits of using multiple representation forms as a strategy to teach domains-specific topics and support a deeper, more abstract understanding of difficult content are well known in mathematics and science education. In physics, for example, the authors in [11] point out that "Many concepts, processes or relations can be comprehended much more quickly when some kind of picture is provided because pictures are able to show at once what would take much longer to be described with words or demonstration experiments. Furthermore, students are able to visualize the rather abstract contents of physics topics being taught such as with the block and tackle. Moreover, when using multiple sources of information, learners are able to choose those sources with which they prefer to learn, in this case the real tackle or the pictures."

In mathematics, for example, students may explore the relationship between variables, through a graph, a table, or an equation. According to [8], much research on mathematical representation is devoted to the study of discipline-specific content such as numbers, fractions or algebra, functions and graphs, etc. These studies look at how students navigate and interact with representations in order to get a deeper mathematical understanding. Students may, for example, use one representation in order to explore another.

There are, thus, discipline-specific strategies for how representations may impact learning and where multiple representations help students discern underlying ideas or concepts that may otherwise be difficult to grasp.

In this paper, we argue that the same benefits can be derived from multi model strategies, where the didactic focus is placed on the relationship and interaction between mathematical and computational models (and their multiple representations), and that these are necessary for the discipline-specific knowledge domains of science education in the computational age.

In other words, we propose a didactic model for teaching science (Physics, Chemistry, Biology) from knowledge and strategies applied in the mathematical and computational modelling process of real (Physics, Chemistry, Biology) phenomena/problems.

We describe the development of the model and theory-base for the introduction of a set of computing concepts as a cross-subject competency in upper secondary courses in sciences through an example of Physics, see Section 2.1 and present one more example of applying the didactic model in Chemistry, see Section 2.2. It is worth mentioned that we have had the opportunity to put the model in practice in more examples from different subjects in upper secondary courses in sciences, for example, Astronomy, Mathematics and Informatics. Some preliminary conclusions are presented in Section 5.

The didactic model was designed and put into practice through a research project titled: "The Paradigm of Computational Sciences and Education", carried out in the period 2021-2023, by researchers of education and informatics, and High School teachers of Physics, Chemistry and Astronomy.

The research problem was defined in the project through a problematisation of the introduction of Computer Science as a basic science in education. Through the cycles of analysing, planning and testing didactic sequences for specific contents, certain critical areas were identified, tested, modified and modelled. In particular, notions of both mathematical and computational models and the relationship between them emerged both as a teaching problem and a potential didactic model. As these were made explicit in the process, teachers and researchers could identify which elements of computing play a relevant role in each didactic sequence and imply a more reflective use of the computer. In this sense we agree with other authors, for example [1, 5, 20], about the potential of fundamental ideas to group the central and far-reaching concepts of informatics, allowing also to distinguish between knowledge, competences, and skills in different approaches. The resulting didactic model organises teaching of com-

putational modelling of a phenomena or problem by bringing to the forefront mainly two fundamental ideas:

1. Algorithms interact with data to solve algorithmic problems (mathematical model).
2. Programs implement algorithms and data in a form that can be executed on a computer (computational model).

For implementing the computational model, the pure functional programming language MateFun was used. The language was designed by researchers at the Institute of Computing (InCo) of the Faculty of Engineering (FING) UDELAR and is aimed at learning mathematical functions [17, 18]. Therefore, its syntax was designed to be minimal and as close as possible to the notation used in mathematics. The idea is that the familiar notation would allow mathematics and other teachers learn the language rather quickly, as well as highlight its relationship with the underlying mathematical concepts, to be discerned by students. Its design began in 2017 and its didactic applications in mathematics courses date back to 2018. From 2021, MateFun was introduced in projects on the teaching of other scientific disciplines, proving to be an especially appropriate language, given that mathematics is the language of science. MateFun's features make it easy to develop the competence of computational modeling of problems, based on mathematical models of their solutions. In this way, teachers of each scientific discipline can introduce their students to basic concepts of computing and reduce the gap between their classroom practices and the paradigm of computational sciences where scientific work is based on three fundamental pillars: computing, theory and experimentation [12].

The MateFun language can be accessed through a web integrated programming environment[3] that allows program management, programming, program execution, and visualisation of graphics, figures, and animations.

## 2 Computational sciences in the classroom

We locate our work within the didactics of informatics, where we have been developing 'close-to-practice' research projects [21], aiming to connect professional knowledge and research through teacher-researcher collaborations in didactic analysis and modelling [3]. As part of the didactic analysis following the methodology that we adopted in the project[4], described in detail in the first example below, all teachers participated in the design of a didactic sequence for the selected topic in their curriculum unit.

### 2.1 An example of physics

For physics, the process began, and evolved through the following problem:

[3] `https://www.fing.edu.uy/proyectos/matefun/#/en/login`
[4] "The Paradigm of Computational Sciences and Education", see Section 1

*How can we calculate and represent the electric field created by any positive charge q at a point located at any distance d from the given charge?*

This problem served as the focal point for analysis as to where each step of abstraction, generalisation and verification could and should be.

At a certain point, teachers and researches agreed that the problem as it is formulated in the statement covers two problems: calculating the value of the electric field and representing the vector that it generates at a point located any distance d from the given charge.

For the first problem, physics teachers calculated the real value for several cases and *implicitly* converting the units, using the formulas in Figure 1. The units are provided by the Coulomb constant k that, according to the International System of Units, uses the Coulomb charge unit (C), the unit of meters for the distance and the Newton (N) unit for force.

Only when the physics teachers entered in conversation with the informatics researchers did it become clear that this was an issue, since from a computational perspective, the data and the conversion of units have to be *explicitly* expressed as data structures and algorithms. At this point, one core element of the model arose: each physics data item must be expressed as an n tuple of the real value and **the corresponding units**. If we call UCharge and UDistance to the units of charge and distance respectively, the statement becomes:

*How can we calculate and represent the electric field created by any positive charge (q, UCharge) at a point at any distance (d, UDistance) from the given charge?*

$$E = \frac{k.q}{r^2} \qquad \mathrm{k} = 9 \cdot 10^9 \frac{N \cdot m^2}{C^2}$$

(a) Equation (b) Constant

Fig. 1: The equation calculates the electric field E produced by a charge q at a point located a distance r from the charge.

The problem question was thus reformulated, as teachers worked to express the problem as an algorithmic problem, that is, specify the input and output data sets as defined in [9]. Several cases used by physics' teachers determined a set of input data of the form ((charge, units), (distance, units)), and for each of them, the output data (electric field, units) were calculated using the equation of the Figure 1. The sets are shown in the first and third columns of Table 1. The conversion of units is not included in the table, but for each case the units

were converted accordingly, worked with the students on the whiteboard and then designing a general function and a program as shown in Figure 2.

**From the equations to a function solving the general problem:** Table 1 shows how the repetition of solving several specific cases groups the input data in the set that forms the first column of the table and the output data in the set of the third column. Each specific case is solved with the equation, where the operations are the same for all cases. This repetition of the operations on the elements of the input data set to generate the output data set makes it possible to naturally introduce the solution of the general case, where the set of equations is transformed into the function, that is, the algorithm. The last row of the table expresses the first step to the mathematical model: a function that **for any pair** (charge, distance) returns the corresponding element of the output data set (the electric field with its units). The next step is to define it as a mathematical function, not included for reasons of space, but which is similar to the function defined in MateFun starting from line 17 of the Figure 3.

Mathematical modelling plays a clarifying role in the process of understanding the problem and a solution. Using a strong typed programming language as MateFun, the relationship between the two models (mathematical and computational) becomes straightforward.

Table 1: From equations to function

| **input data** | **algorithm** | **output data** |
|---|---|---|
| ((3,NanoCoulomb), (2,Cm)) | equation | (67500,Newton,Coulomb) |
| ((3,NanoCoulomb), (0.3 Mts)) | equation | (300,Newton,Coulomb) |
| ((10,MicroCoulomb), (1, Mts)) | equation | (90000,Newton,Coulomb) |
| ((5,MicroCoulomb), (10,Cm)) | equation | (4500000,Newton,Coulomb) |
| ((0.5, Coulomb), (800, Cm)) | equation | (70312500,Newton,Coulomb) |
| ... | equation | ... |
| **for all** (charge, distance) | **function** | (EField, Units) |

**From function to program:** Although a mathematical function expresses the general solution of a problem, the strength of computing in science education is revealed in all its magnitude when the algorithm is transformed into a program that can be executed by a computer for several cases. The process of computational modelling has a great impact, both for teachers and students, where they can experience their solutions in action [15]. A program in MateFun is a list of definitions of **sets** and **functions**. Figure 2 shows the definitions of the data types for the units of charge, distance and force (they can be expanded with new units) and the MateFun functions to convert the units. Figure 3 shows the electrical constant defined as a function that has no input data (empty domain) and that returns a real value in Newton and Coulomb. The same figure shows the

```
set UCharge   = { Coulomb , MicroCoulomb, NanoCoulomb }
set UDistance = { Km, Mts, Miles, Cm }
set UForce    = { Newton }
convertUnitCharge :: R X UCharge -> R X UCharge
convertUnitCharge (c,u)
        = { (c * 1e-9, Coulomb) if  u == NanoCoulomb
          { (c * 1e-6, Coulomb) if  u == MicroCoulomb
          { (c, Coulomb)
convertUnitDistance :: R X UDistance -> R X UDistance
convertUnitDistance (d,u)
        = { (d * 0.01, Mts)    if u == Cm
          { (d * 1000, Mts)    if u == Km
          { (d * 1609.34, Mts) if u == Miles
          { (d, Mts)
```

Fig. 2: Definitions of data types for charge, distance and force units. Definitions of unit conversion functions.

definition of the function *electricField* that computationally models the solution of the problem of calculating the electric field produce by a charge in a point situated a certain distance from the charge (first sub problem, see Section 2.1). The problem of representing the vector of the electric field at a point located at a distance d from the charge (see Section 2.1) was analysed with the same methodology used in the first problem, that is, solving several points for specific distances and a charge of 3 NanoCoulomb, and then constructing a general solution, where the points are elements of a list. Figure 4 shows the definition in MateFun of the function to draw the vectors (omitting the auxiliary functions) and the expression to apply the function to a charge of 3 NanoCoulomb at a scale of 0.0001, and to the list [(2,0),(1,1),(0,2.5),(-1,-1)] of the coordinates of the points on the plane. Two relevant issues arose here: the need to use a scale to represent the electric charge and to determine that the points were located in the plane. Discussing these issues, made evident and experiential ideas about the need for rigour in computational modelling, that had not yet been fully assimilated by the physics teachers up until the final stage of the didactic modelling process. At the same time, the issues brought by physics' teachers cast to light some important ideas for the computer science researches about how to improve the MateFun language: including a physics library where teachers can easily use predefined functions to write their expressions.

Figure 5 shows the graphical representation on the left side and the expression executed on the right side of the MateFun environment. It is worth noting that in the case where the charge is negative, the same expression is used with the negative value and the result is that the direction of the electric field vectors points towards the charge. The teaching sequences were put into practice in the classes, which were filmed for later analysis. The analysis of the first results

```
15 constElec :: () -> (R X UForce X UDistance X UCharge)
16 constElec () = (9000000000, Newton, Mts, Coulomb)
17 elecField :: ((R X UCharge) X (R X UDistance)) -> (R X
   UForce X UCharge)
18 elecField (charge, distance) = (constElec ()!1 * charge1
   !1 / (distance1!1 * distance1!1), Newton, Coulomb)
19           where charge1 = convertUnitCharge (charge)
20                   distance1 = convertUnitDistance (
   distance)
21 {- Test
22 >electricField((3,NanoCoulomb), (2,Cm))
23 (67500,Newton,Coulomb)
24 >
25 -}
```

Fig. 3: Definitions of the Coulomb constant and the electric field function.

```
26 drawField :: (R X TCharge) X R X (R X R)* -> Fig
27 drawField (charge,scal,pts) = draw(charge, pts,
   calcDestinations(pts,ds,calcFields(charge,scal,ds)))
28               where ds = calcDistances(pts)
29 {- Test
30 Elecfield>drawField((3,NanoCoulomb),0.0001,[(2,0),(1,1)
   ,(0,2.5),(-1,-1)])
31 -}
32
```

Fig. 4: MateFun function for the graphic representation of the electric field.

allowed us to develop a didactic model with the objective of introducing students' learning about effective computational models and its relationship with mathematical models. Further improvements no worked in these instructional instances are included in the Section 4 and in Section 5 we present some criteria of the analysis and some results of the collected data.

This year we had had the opportunity of applying the model in three instructional instances of three hours and a half each. Four groups of students and teachers of Chemistry, Astronomy, Physics and Visual Communication participated in the activities that were organized and supported by PEDECIBA and ANEP[5].

[5] PEDECIBA(Programa de Desarrollo de las Ciencias Básicas https://www.pedeciba.edu.uy/en/) ANEP (Adminsitración Nacional de Educación Pública https://www.anep.edu.uy/).

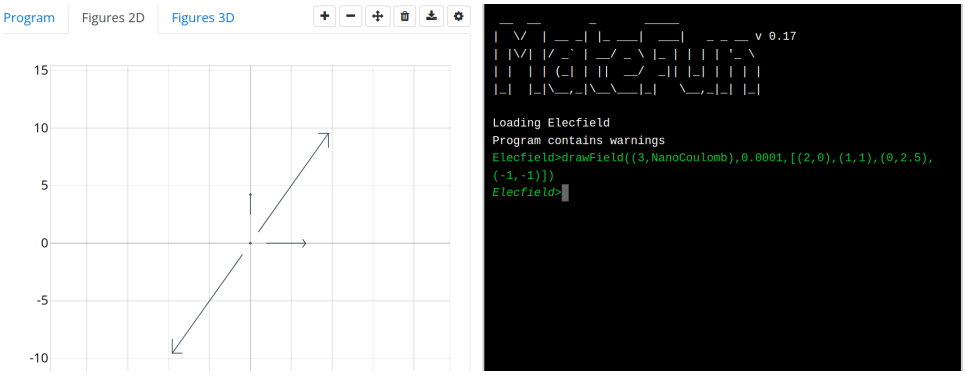



Fig. 5: Graphical representation of electric field vectors in MateFun environment.

Each group presented the final work in a common event that took place in November 8th, to an audience of researchers in informatics and education and High School teachers.

In the following Section we describe the work done with the Chemistry group.

### 2.2 An example of chemistry

Due to time constraints, the basic tools for using MateFun were introduced in the first hour of the first instructional instance. The implementation of the functions was introduced as they were defined mathematically, making explicit the concepts that are usually implicit. For example, describing the domain and co domain of functions as the Cartesian product between R and the sets of units.

**First instructional instance.** The teacher of Chemistry brought the following statement of a problem that she had been worked with the students:

> *Commercial nitric acid has a concentration of 69%m-m and a density of 1.42 g/mL. Calculate its concentration expressed in g/L and its molarity.*

As in the example of Physics, the core element of the model arose: each chemistry data item must be expressed as an n tuple of the real value and **the corresponding units**. The sets of units are described in the Figure 6 and the statement formulated as an algorithmic problem is:

> *Given a quantity in grams of solute in grams of a solution, and the density of the solution in grams per milliliter, return the concentration of the solution expressed in grams per liter and its molarity in moles per liter.*

For designing a solution, the problem is divided into two sub problems: 1) calculating the concentration and 2) calculating the molarity.

In the first instructional instance the first sub problem was analyzed an solved as follows: for the concentration of the solution in grams per liter we need to know what volume is the grams of solution, that is, to convert from mass to volume (that is why the density that relates both things is input data). That is to say, "given a quantity in grams of the solution (input data) and the density of the solution in grams per milliliters (input data), return the volume of the solution in milliliters". Once the volume in milliliters is calculated, a simple rule of three gives the requested concentration in liters (output data of the sub problem). The solution in MateFun is shown in the Figure 6.

```
33 {- We define the sets of units -}
34 set UVol = { Liter, Milliliters }
35 set UMass = { Grams, Milligrams }
36 set UMolWeight = { GramsPerMole }
37 set UMoles = { Moles }
38 set UMolarity = { MolesPerLiter }
39 {- Program: We define the function to solve the
   subproblem using the density formula. -}
40 getVolInML :: (R X UMass) X (R X UMass X UVol) -> R X
   UVol
41 getVolInML (mass, dens) = (mass!1 / dens!1 , Milliliters
   )
42 concentrationInLiters :: (R X uMass) X (R X UMass) X (R
   X UMass X UVol) -> (R X UMass X UVol)
43 concentrationInLiters (gramsSolution, gramsSolute, dens)
    = (gramsSolute!1 * 1000 / volu!1 , Grams, Liter)
44                          where volu = getVolInML (
   gramsSolution,dens)
```

Fig. 6: Calculating concentration in liters

**Second instructional instance** For the second sub problem the input data is the mass in grams of the solution, the mass in grams of the solute, its molecular weight and its density, and the output data is the molarity of the solution (nitric acid). The formula for the molarity is shown in the Figure 8, where the number of moles is calculated using the atomic mass taken from the periodic table of chemical elements.

**Third instructional instance** According to our model, once a solution is obtained and testing, it should be analized to find aspects that can be improved, for instance from the point of view of getting a more compact and efficient code. In this case, we encourage students to pay attention to the fact that the local

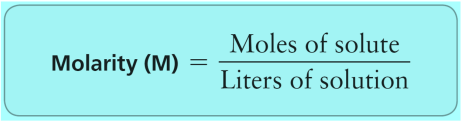
definition of the variable 'volu' using the function 'getVolInML' is used in both sub problems. They are encouraged to define a function that takes the grams of solution, the grams of solute, the density in grams per milliliter of the solution (input data), the molecular weight of the solute (data taken from the periodic table of chemical elements) and returns a pair with the concentration of the solution in grams per liter (sub problem 1) and the molarity of the solution in moles per liter (sub problem 2). In this way, the variable volu is defined once and used in both sub programs. The program is described in the Figure 9 The volume of the solution is calculated with the getVolInML function of the first sub problem and multiplied by 1000 to obtain the volume in liters. The program is described in Figure 7.

```
45 {- Functions to calculate the number of moles of solute
   (in moles) and the molarity. -}
46 numberMoles :: (R X UMass) X (R X UMolWeight)  -> R X
   UMoles
47 numberMoles (massSolute, molWeight) = (massSolute!1 /
   molWeight!1, Moles)
48 molarity :: (R X UMass) X (R X UMass) X (R X UMolWeight)
    X (R X UMass X UVol) -> (R X UMolarity)
49 molarity (massSolution, massSolute, molWeight, dens) = (
   m!1 / (volu!1/1000), MolesPerLiter)
50                     where m = numberMoles (massSolute,
   molWeight)
51                           volu = getVolInML(massSolution,
    dens)
52 {- Concrete case:
53 molarity((100,Grams), (69,Grams),(63,GramsPerMole)
   ,(1.42,Grams,Milliliters))
54 (15.5524,MolesPerLiter)
55 -}
```

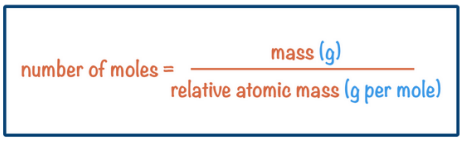
Fig. 7: Calculating the molarity

## 3 Related work

There is a great deal of consensus about the fact that Computer Science Education (CSE) is becoming more and more relevant and that most other sciences use digital technology to present their concepts and elaborate information (simulation of physical phenomena, DNA sequencing and analysis, or manipulation of abstract geometrical objects, and so on). However, the relationship between CSE and science education is interpreted in different ways. In this section, we refer to




(a) Molarity Formula




(b) Number of Moles

Fig. 8: Formulas for the second chemistry sub problem.

two that we compare with our approach. One of them is the one that covers works that support the need to introduce Computational Thinking (CT) in science education [2,10], in the understanding that the skills of a computer scientist can be transferred to problem solving in other domains, an idea introduced by [13]. This approach is being abandoned, as no evidence has been found that such a transfer is possible. The other approach is to consider CT as a set of skills, such as abstraction, decomposition, etc., that arise when solving algorithmic problems, bypassing the stage of implementing solutions in a programming language. This vision had great resonance from the initial definition of Wing that introduces the idea that CT can be developed from the domain of general competences and *without the use of the computer* [10]. This approach to basic computing presents low entry requirements, which allows teachers, professors, and students to have their first approaches to computing, without having to handle too many computing concepts and without requiring prior programming knowledge [7]. The academic community has expressed concern for years about the way in which this limited vision reduces computing to a few basic components, which, in addition to being common to all sciences, obfuscate the algorithm-machine relationship that is the basis of computing as a discipline [5, 14].

The other approach to science education in the digital age is to use different software specialized in certain disciplines, such as GeoGebra in mathematics. In these cases, the introduction of computer science concepts in science education is not addressed either, since most of the programs are provided by the tools and the focus is on visualization and simulation.

On the other hand, Physics VPython Open Source in Physics[6] uses VPython a language that requires certain knowledge of imperative programming but is disconnected from the mathematical model of the solutions; for instance, it does not require indicating the signature of functions.

[6] see for example https://matterandinteractions.org/wp-content/uploads/2016/07/Chapter1-InteractionsandMotion.pdf

```
{- Function that solves the main problem by returning a
pair with the solution of the each subproblem -}
concentrationMolarity :: (R X UMass) X (R X UMass) X (R
X UMass X UVol) X (R X UMolWeight) -> ((R X UMass X UVol
) X (R X UMolarity))
concentrationMolarity (gramsSolution, gramsSolute, dens,
 molWeight) = ((gramsSolute!1 * 1000 / volu!1 , Grams,
Liter) , (mols!1 / (volu!1/1000), MolesPerLiter))
                    where mols = numberMoles (gramsSolute
, molWeight)
                          volu = getVolInML(gramsSolution
, dens)
{-  Test of the first sub program for the specific case:
>concentrationMolarity((100, Grams), (69, Grams), (1.42,
 Grams, Milliliters), (63, GramsPerMole))!1
(979.8000,Grams,Liter)
Chemistry>
{-  Test of the second sub program for the specific case
:
>concentrationMolarity((100, Grams), (69, Grams), (1.42,
 Grams, Milliliters), (63, GramsPerMole))!2
(15.5524,MolesPerLiter)
Chemistry>
-}
```

Fig. 9: Solving both parts of a problem from chemistry

Finally, we note that there are contributions of Functional Programming to science education, especially in Mathematics (for example "Discrete mathematics using a computer" by Cordelia Hall and John O'Donnell) and in Physics (for example "Physics and Functional Programming" by Scott N. Walck).

Although all the approaches mentioned are valuable, we would like to highlight some strengths of our didactic model: firstly, it arises from the joint work of education professionals (teachers), researchers in specific didactics, and researchers in computer science (see Section 1, based on the analysis of problems and exercises brought by teachers. Secondly, the functional programming language MateFun was specially designed to explain the relationship between the mathematical modelling of problems and computational modelling of their solutions, with a simple syntax that is easily accessible to teachers and students through the web environment. Finally, the didactic model is based on theoretical rationale: the fundamental ideas of computing [1, 3, 20] and an epistemological model on the construction of knowledge about data structures, algorithms and programs, which we have developed over the years [16, 19].

## 4 Further work

As mentioned in Section 2.2, once a primary solution is written and executed, the students are encouraged to analyse it and discuss ways of obtaining more compact code and/or more general and efficient programs. This stage requires spending more time with students and teachers. We are designing teaching materials and planning workshops to be held next year in different educational institutions throughout the country. An important part will be a more detailed study of MateFun and the analysis of programs that can be improved. As an example we include in the Figure 10 a new solution of the physic problem described in Section 2.1.

As mentioned in 2.1 we are implementing MateFun libraries for the different scientific disciplines to facilitate teachers' work. For instance, the constants, the functions to unit conversion, the periodic table of chemistry elements will be include in the libraries.

## 5 Conclusions

In [6], page 159, the authors describe how the development of computer science produced a paradigm shift that enabled many new scientific discoveries. Scientists in all fields have found that computer science brings a new method of doing science, that is added to the classic methods of theory and experiment. In education, in the best cases, some basic concepts and rules for mainly imperative programming are introduced and that in science education, computer science is generally absent or it is reduced to using some digital tools.

The authors affirm that education efforts are necessary to diminish the gap between scientific work in the new paradigm and what students are taught in classrooms, in all educational levels. We agree with authors' opinions and add that the gap is specially serous in High School, where teachers practice is usually disconnected from research.

Internationally, we have seen efforts that greatly expanded the computer education curriculum to encompass multiple areas and key computing competencies. In 2022, for example, the European Commission (EC) published the report: 'Informatics education at school in Europe' which provides a comprehensive comparative analysis of Computer Science Education (CSE) as its own discipline in primary and secondary education in thirty-nine education systems [4].

In its analysis of the integration of computing in different education systems, the authors review various frameworks with diverse learning objectives and then operationalise ten competency areas. These ten areas 'aim to capture the recurrent content in existing competency frameworks and therefore provide a general understanding of the possible content of informatics subjects' (see page 42 of [4]. The Eurydice report provided us with the following definition: 'Computational modelling and simulation help people to represent and understand complex processes and phenomena. Computational models and simulations are used, modified, and created to analyse, identify patterns, and answer questions of real phenomena and hypothetical scenarios'.

We propose a didactic model for teaching science from knowledge and strategies applied in the mathematical and computational modelling process of real phenomena/problems. We describe the development of the model for the introduction of a set of computing concepts as a cross-subject competency in upper secondary courses in sciences through an example of Physics, see Section 2.1 and present one more example of applying the didactic model in Chemistry, see Section 2.2.

The choice of a functional programming language as MateFun facilitate the task by allowing the mathematical model to be implemented computationally in a direct way, which also coincides with the introduction of the fundamental ideas numbered in Section 1. For instance, formulating the statement as an algorithmic problem forces us to pay attention to the input and output data and structure them as n tuples, which is traditionally treated implicitly, but in a computational model must be explicitly implemented. MateFun's features make it easy to develop the competence of computational modelling of problems, based on mathematical models of their solutions. An example of the contribution to education is Matefun's simple syntax that requires defining functions including domain and co domain (like almost all functional programming languages and unlike others).

We have experienced that students find it a motivating complement to go through the stages of our teaching model. At the event to present the final works mentioned at the end of 2.1, it was surprising that in just three instructional instances they were able to achieve solutions that, although they could be improved, showed the first steps towards developing modelling skills for problems in different scientific disciplines.

## References

1. Bell, T., Tymann, P., Yehudai, A.: The big ideas in computer science for k-12 curricula. Bulletin of EATCS, 1(124). (2018)
2. C., O., M., T.S.: Computational thinking in introductory physics. The Physics Teacher 58(4) pp. 247–251 (2020)
3. Cabezas, M., da Rosa, S.: Modelado didático para ideas fundamentales en computación. Proceedings of The 51 SADIO Conference, Simposio Argentino de Educación en Informática (SAEI 2022) (2022)
4. Commission, E., Education, E., Agency, C.E.: Informatics education at school in europe. `https://eurydice.eacea.ec.europa.eu/publications/informatics-education-school-europe` (2022)
5. Denning, P., Tedre, M.: Shifting Identities in Computing: From a Useful Tool to a New Method and Theory of Science. In Hannes Werthner and Frank van Harmelen, Eds. Informatics in the Future, Proceedings of the 11th European Computer Science Summit (2015)
6. Denning, P., Tedre, M.: Computational Thinking. Cambridge, MA : The MIT Press (2019)
7. Denning, P., Tedre, M.: Computational thinking: A disciplinary perspective. Informatics in Education. **20(3)**, 361–390 (2021). `https://doi.org/10.15388/infedu.2021.21`

8. G., G.: Mathematical representations. In book: Encyclopedia of Mathematics Education. Available from: https://www.researchgate.net/publication/289514617_Mathematical_Representations [accessed Nov 13 2024]. (2014)
9. Harel, D., Feldman, Y.: Algorithmics - The Spirit of Computing. Addison-Wesley Publishers Limited 1987, 1992, Pearson Education Limited 2004 (2004)
10. Lodi, M., Martini, S.: Computational thinking, between papert and wing. Science & Education volume, 30 pp. 883—-908 (2021)
11. M., O., A., S., E., F.: Chapter 1: Multiple Representations in Physics and Science Education – Why Should We Use Them? In Springer International Publishing. D.F. Treagust et al. (eds.), Multiple Representations in Physics Education, Models and Modeling in Science Education Vol. 10 (2017)
12. M., R., F., W., D., K., A., S., T., L.: Research advances by using interoperable e-science infrastructures. Cluster Computing 12(4), DOI 10.1007/s10586-009-0102-2 pp. 357–372 (2009)
13. Papert, S.: Mindstorms: children, computers, and powerful ideas. Basic Books, Inc. New York, NY; 1980. ISBN:0-465-04627-4 http://dl.acm.org/citation.cfm?id=1095592 (1980)
14. da Rosa, S.: Piaget and computational thinking. CSERC '18: Proceedings of the 7th Computer Science Education Research Conference pp. 44–50 (2018)
15. 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)
16. da Rosa, S., Viera, M., García-Garland, J.: A case of teaching practice founded on a theoretical model. Proceedings of The 13th International Conference on Informatics in Schools (2020)
17. da Rosa, S., Viera, M., García-Garland, J.: Mathematics and matefun, a natural way to introduce programming into school. (2020)
18. da Rosa, S., Viera, M., García-Garland, J.: Training teachers in informatics: a central problem in science education. Proceedings of The 50 SADIO Conference, Simposio Argentino de Educación en Informática (SAEI) (2020)
19. da Rosa, S., Gómez, F.: The construction of knowledge about programs. Proceedings of PPIG 2022 - 33rd Annual Workshop p. 1–8 (2022)
20. Schwill, A.: Computer science education based on fundamental ideas. Proceedings of the IFIP TC3 WG3.1/3.5 joint working conference on Information technology: supporting change through teacher education pp. 285–291 (1997)
21. Wyse, D., Brown, C., Oliver, S. & Poblete, X.: Education research and educational practice: The qualities of a close relationship. British Educational Research Journal, 47(6) pp. 1466–1489 (1993)

```
70  set UCharge   = { Coulomb , MicroCoulomb, NanoCoulomb }
71  set UDistance = { Km, Mts, Miles, Cm }
72  set UForce    = { Newton }
73  set Charge   = { c in R X UCharge }
74  set Distance = { d in R X UDistance }
75  set Force    = { f in R X UForce }
76  convertUnitCharge :: UCharge X UCharge -> R
77  convertUnitCharge (ci,cf)
78          = { 1e-9 if  (ci == NanoCoulomb,  cf == Coulomb)
79            { 1e-6 if  (ci == MicroCoulomb, cf == Coulomb)
80            { 1e9  if  (ci == Coulomb,      cf ==
    NanoCoulomb)
81            { 1e6  if  (ci == Coulomb,      cf ==
    MicroCoulomb)
82            { 1e3  if  (ci == MicroCoulomb, cf ==
    NanoCoulomb)
83            { 1e-3 if  (ci == NanoCoulomb,  cf ==
    MicroCoulomb)
84            { 1
85  convertCharge :: Charge X UCharge -> Charge
86  convertCharge (c,u) = (c!1 * convertUnitCharge(c!2,u), u
    )
87  convertUnitDistance :: UDistance X UDistance -> R
88  convertUnitDistance (di, df)
89          = { 1e3           if (di == Km,    df == Mts)
90            { 1e-3          if (di == Mts,   df == Km)
91            { 1609.34       if (di == Miles, df == Mts)
92            { 6.2137e-4     if (di == Mts, df == Miles)
93            { 1e5           if (di == Km, df == Cm)
94            { 1e-5          if (di == Cm, df == Km)
95            { 1e2           if (di == Mts, df == Cm)
96            { 1e-2          if (di == Cm, df == Mts)
97            { 160934        if (di == Miles, df == Cm)
98            { 6.2137e-6     if (di == Cm, df == Miles)
99            { 1
100 convertDistance :: Distance X UDistance -> Distance
101 convertDistance (d,u) = (d!1 * convertUnitDistance(d!2,u
    ), u)
102 k :: () -> R X UForce X UDistance X UCharge
103 k () = (8.9875e9, Newton, Mts, Coulomb)
104 set ElectricField = { e in R X UForce X UCharge }
105 computeElectricField :: Charge X Distance ->
    ElectricField
106 computeElectricField (q, r) = ((k()!1 * qn!1) / (rn!1 ^
    2), k()!2, k()!4)
107     where
108         qn = convertCharge(q, k()!4)
109         rn = convertDistance(r, k()!3)
110
```

Fig. 10: A more compact and general function for compute the electric field.