Computer tools in education, 2019
№ 2: 55–64
http://cte.eltech.ru
doi:10.32603/2071-2340-2019-2-55-64

MATHEMATICAL MODELING AND PROGRAMMING IN SCIENCE EDUCATION

Weigend M.[1,2], Dr., mw@creative-informatics.de

[1]Holzkamp-Gesamtschule, 2, Willy-Brandt-Str., 58453, Witten, Germany
[2]Westfälische Wilhelms Universität, 2, Schlossplatz, 48149, Münster, Germany

Abstract

Using mathematical models to represent aspects of physical reality is an essential activity in science and science education. This contribution discusses four approaches of using computer programming and mathematical models in classroom activities:

1. Mathematical models, found in the textbook, are used as a basis for computer programs. Students, when creating useful interactive python programs calculating concentrations or pH-values, experience similar intellectual challenges as in solving traditional text book problems.

2. Scratch-animations simulating physical or chemical systems simulation can be specifically designed to check the validity of given mathematical models.

3. A computer-related challenge is to design a simulation (like gas diffusion in a closed system with two phases) that might be a basis for discovering a mathematical model (like Henry's law) or just an element of a mathematical model.

4. Using sensor technology and a Raspberry Pi, students create a computer program that automatically visualizes the observed system behaviour (like changes in gas concentrations) in order to find a mathematical model.

Keywords: *Science education, Python, Scratch, Programming.*

Citation: M. Weigend, "Mathematical Modeling and Programming in Science Education," *Computer tools in education*, no. 2, pp. 55–64, 2019; doi:10.32603/2071-2340-2019-2-55–64

1. INTRODUCTION

Models of the physical reality that are taught in high school science lessons are represented by mathematical equations. Well known examples from physics are Einstein's equation $E = m \cdot c^2$ representing the idea that energy can be transformed to matter and vice versaand $E = h \cdot f$ representing the dual model of light (discrete photons with energy E versus electromagnetic waves with a frequency f).

There exist special digital tools for performing simulations and discovering and exploring models. They can be grouped in two types:

- Open modelling tools like Insightmaker (https://insightmaker.com/) allow students to create dynamic models of physical systems and study their behaviour.
- In virtual laboratories, students can simulate experiments that are too expensive, too time-consuming or too dangerous in reality [3].

Specialized digital tools have great potential for learning. However, they have certain limitations:

- Students use highly specialized software-tools only seldom. Still they have to acquire technical knowledge to be able to handle the tool. And part of this knowledge is not transferable and not related to any educational goal.
- A virtual laboratory is a surrogate for reality and hides the underlying mathematical models.

Programming projects — using a general purpose programming language like Python or Scratch — are different from classroom activities with specialized digital tools:

- They imply creating useful digital artifacts and have a stronger focus on engineering and designing than on researching. “Learning by creating” is the educational paradigm of “Constructionism” ([1, 2]).
- The students need to develop general informatics competences before and during the project. This takes time and may require some “professional support” by a computer science teacher. However, the acquired computer-related knowledge is deeper and more general. “Computational thinking” is more and more considered as a competence that everybody should develop — not just professional programmers [6].
- Programming challenges can be designed in a way that mathematical modelling is specifically involved.

This contribution discusses four approaches of integrating computer programming and mathematical modeling in *science education*: 1) Using mathematical models as a basis for simple interactive Python programs. 2) Developing animations simulating physical or chemical systems in order to check the validity of given mathematical models. 3) Designing computer simulations in order to discover mathematical models. 4) Creating computer programs that automatically visualize sensor data and support checking mathematical models.

2. APPLYING MATHEMATICAL MODELS TO CREATE COMPUTER PROGRAMS

High school science textbooks contain exercises that require algebraic transformations and calculation. The educational idea is to comprehend the mathematical models by applying them.

Example from a German Chemistry textbook [4]: Calculate the pH of these solutions:

- Sodium hydroxide solution, $c_0(NaOH)$ = 1 mol/L
- Ammonia solution, $c_0(NH_3)$ = 0.2 mol/L

The solution of this task requires these activities:

- Find relevant mathematical models: $pH = -log_{10}(c(H_3O^+))$ and $pH + pOH = 14$.
- Do algebraic transformations and solve equations.
- Look up chemical constants (like $pKB(NH_3)$)in a table.
- Use correct wording, in particular write correct units like mol/L for concentrations.

In contrast to conventional pencil and paper textbook exercises, a programming task requires to develop a digital artifact, that is useful for other people. Example: Write a Python program that asks for the concentration of sodium hydroxide ($NaOH$) in water and then calculates the pH of the solution. That implies basically the same activities as in finding the solution of a textbook problem, but they are now part of a design process. The programmer must create verbal system responses that are precise and well understandable and uses scientific vocabulary and units because they are essential, when asking for input data. Example “Input the concentration of NaOH in mol/L.” For programming beginners, the first approach would be an interactive program, following an input-processing-output pattern.

```
from math import log10
print("Enter the concentration of the NaOH solution.")
x = input("c(NaOH) in mol/L: ")
c = float(x)
if c < 10**(-7):                                    #1
    c = 10**(-7)
pH = 14 - (-log10(c))                               #2
print("pH", round(pH, 1))
```

Example dialog:

```
Enter the concentration of the NaOH solution.
c(NaOH) in mol/L: 0.2
pH 13.3
```

The program is so short that beginners can develop it from scratch. Line #2 matches the equation that a student needs to develop when solving the first word problem. The word problem implies calculations for one given concentration, whereas the program should be able to process all concentrations and calculate reasonable results. This makes the program more complex than the single case of a word problem and it requires a deeper elaboration of the chemistry model

The statement in line #2 is reasonable under the assumption that $NaOH$ dissociates completely to sodium ions Na^+ and hydroxide Ions OH^-. Since in neutral water the concentration of hydroxide ions is only 10^{-7} mol/L, in all solutions of practical relevance c(OH-) is approximately equal to $c(NaOH)$. However, the computer program must handle the case that the user inputs a lower concentration, may be even 0 mol/L. The statement in line #1 takes care of this. Here the programmer has used the knowledge that in neutral water pH equals pOH. The equation $pH + pOH = 14$ then implies that pOH = 7. Thus $c(OH^-) = 10^{-7}$ mol/L. In any solution of $NaOH$ in water, the concentration of hydroxide ions may never be below this value. More experienced students may want to develop a version with a graphical user interface (GUI). The advantage is that is easier to use and that it looks more like a regular app in everyday life.

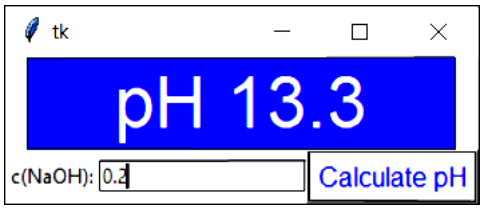

Figure 1. Application window of a Python program calculating pH-values

Obviously, the following Python listing is more advanced. Note that the complete chemistry-related calculation is within a single function definition starting at line #1. So, beginners could take a prototype program text that already implements the GUI and just code this function.

```
from tkinter import *
from math import log10

def calculate():                                    #1
  x = entry.get()
  c = float(x)
  if c < 10**(-7):
```

```
        c = 10**(-7)
    pH = 14 - (-log10(c))
    output = "pH {:.1f}".format(pH)
    resultLabel.config(text=output)

# Widgets
window= Tk()
label = Label(master =window, text = "c(NaOH):")
resultLabel = Label(master=window, width = 10,
                font=("Arial",32), fg="white",
                bg = "blue")

button = Button(master=window,text="Calculate pH",
                  command=calculate,
                  font=("Arial",12), fg="blue" )
entry = Entry(master=window, width=20)

# Layout
resultLabel.pack()
label.pack(side=LEFT)
entry.pack(side=LEFT)
button.pack(side=LEFT)
window.mainloop()
```

3. CHECKING MATHEMATICAL MODELS USING COMPUTER SIMULATIONS

In high school education, mathematical models are often just used for calculations, but not deduced from assumptions. A classic example from kinetics is the equation

$$s = s_0 + \frac{1}{2}at^2. \tag{1}$$

It specifies the location s of a uniformly accelerated object. This equation can be derived from the definition of acceleration $a = \frac{dv}{dt}$ by algebraic rearrangement and integration. Alternatively, it is possible to create a computer simulation, modelling a rocket or a free-falling stone. If position, speed, acceleration and elapsed time are documented one can compare the behaviour of the simulation with the mathematical model. Figure 1 depicts a Scratch project simulating a uniformly accelerated object. To avoid negative numbers, it is not a free-falling object driven by gravity but a rocket car moving from left to right.

The script reveals the modelling: The change of position during a time step dt is determined by speed ($ds = v \cdot dt$) and the change of speed is determined by acceleration ($dv = a \cdot dt$). Command block 1 calculates the position that is expected according to the mathematical model (equation 1). During run time one can watch the readouts on top of the screen and compare the values. Obviously, the values are similar but not equal. The smaller the value of the time step dt the better the numerical approximation. In a real physical experiment [5], the mathematical model can be verified just by observing a real falling object measuring distances and time intervals. Note the difference: In this approach the model is considered as a black box model. It describes the behaviour but does not explain it. Real experiments for checking mathematical models can be dull and time consuming. For example, the Beer-Lambert law is the

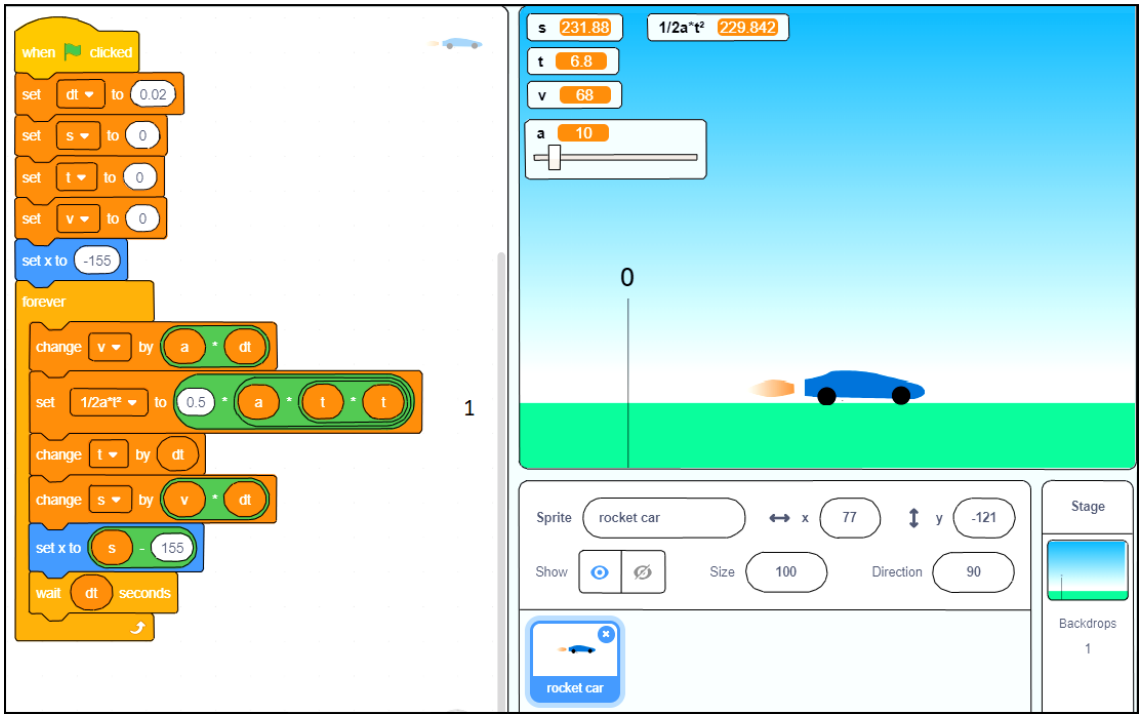

Figure 2. Scratch project modelling an accelerating rocket car

basis of photometry, a method to determine the concentration of a dye solution by measuring the absorbance of monochromatic light of a certain wavelength. The absorbance (extinction) is measured according to equation (2), I0 is the initial light intensity and I1 is the intensity of the light after having travelled through the solution

$$A = lg(\frac{I0}{I1}). \tag{2}$$

According to the Beer-Lambert Law, the absorbance of a dye solution is proportional to the concentration (mols per litre) and path length. The constant ϵ is the molar extinction coefficient, which is a physical property of the analyte and depends on the wavelength of the monochromatic light travelling through the solution

$$A = \epsilon \cdot d \cdot .c \tag{3}$$

The derivation of the Beer-Lambert Law requires knowledge of the calculus that high school students (in Germany) usually do not have. Fig. 3 shows a screen shot from a simple Scratch simulation, illustrating that the absorbance is proportional to pathlength. The green spots on the background represent dye molecules in the solution. Red photons travel from left to right starting at random y-positions and leaving a red track. When a photon hits a molecule, it vanishes. The absorption events are counted. The two readouts on top show the number of photons, which have been absorbed in the first half and the number of photons, which have been absorbed, in the second half. From these numbers the light intensities I0 (initial intensity), I1 (intensity in the middle) and I2 (intensity after the solution) are calculated. It may seem plausible that the absorbency of light is proportional to the pathlength. But it might be disturbing that the logarithm is used to calculate the absorbance. The simulation is not an analytical proof

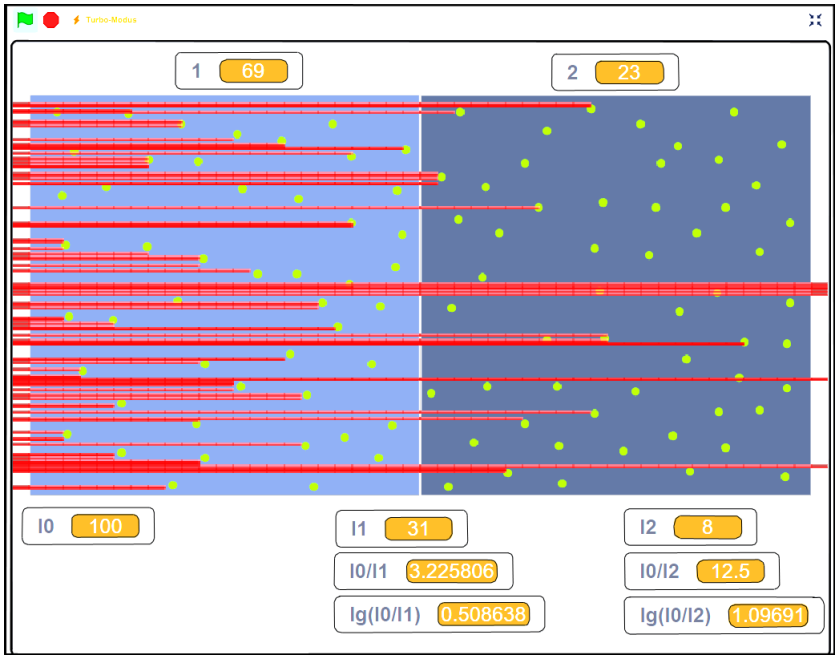

Figure 3. Scratch project illustrating the Beer-Lambert-Law.

that the Beer-Lambert Law is correct. But it demonstrates that it is a logical consequence of the assumptions we have made in the design of the model.

4. CREATING A COMPUTER SIMULATION IN ORDER TO FIND A MATHEMATICAL MODEL

It is very hard and time consuming to discover quantitative scientific laws and create a mathematical model. People can only be creative in a domain they know well. If students are experienced in Scratch programming, they can use this knowledge to design a simulation that might be a basis for a mathematical model.

Task: Consider a closed system with two phases — fluid and gas — like a bottle of mineral water. The water (phase 1) contains solved carbon dioxide as well as the air (phase 2) on top. The carbon dioxide molecules move, when they hit the interface between the two phases, they just change to the other phase, from water to air and vice versa. Assumptions: 1) The CO_2 molecules move with a constant speed on straight trajectories. 2) CO_2 molecules move slower in the water than in the air, since they hit water molecules all the time (the water molecules are invisible).

Challenge: 1) Create the Scratch model. 2) Observe the Scratch model with different numbers of molecules and write a hypothesis (equation). 3) Try to verify the hypothesis using the Scratch model.

Fig. 4 shows a possible implementation with Scratch using clones and message passing. The first script generates several clones. The second scripts defines the motion of each clone (molecule). When a molecule is in the water phase (dark blue area at the bottom), it moves slower than in the gas phase (light blue area on top).

A version of Henry's Law is

$$\frac{c(CO_2(g))}{c(CO_2(aq))} = k. \tag{4}$$

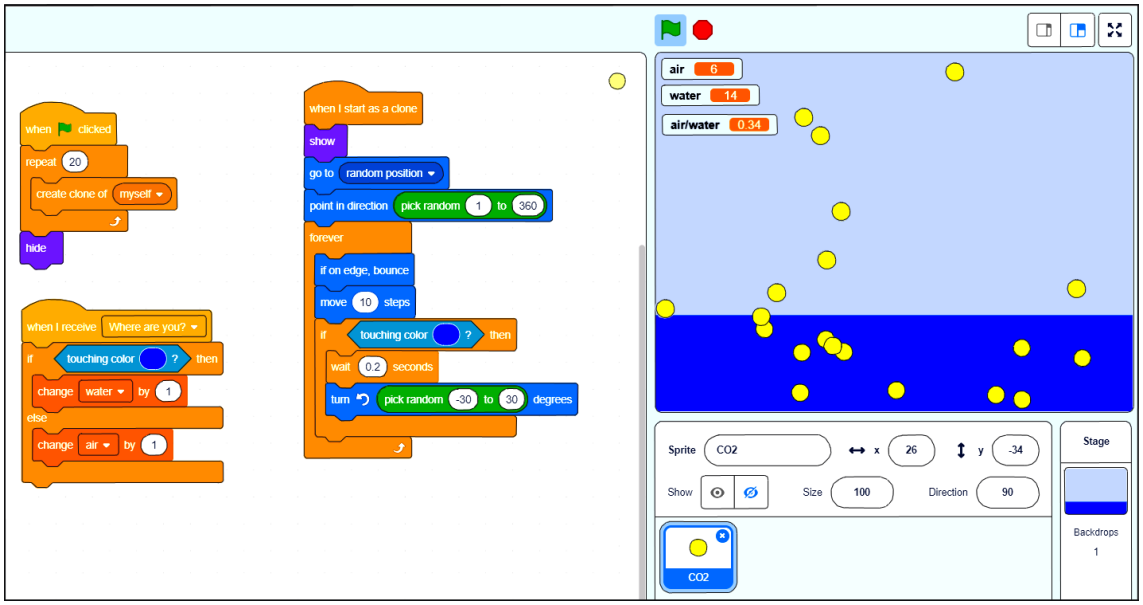

Figure 4. Scratch project simulating gas particles in two phases and checking Henry's Law.

If the volume is constant, the concentration is proportional to the number of molecules. In the background, a script (connected to the stage) is running sending a message to all molecule-clones every four seconds. When a molecule-clone receives the message, it checks its position and increments a global variable. Thus, we have the numbers of molecules in both phases (variables displayed in the upper left corner of the screen). On the screen shot in fig. 4 the third readout on screen displays the quotient of the numbers of "CO_2-molecules" in both "phases" this way checking Henry's Law. Note that this Scratch program is not just a simulation. Additional to representing objects of the physical world it includes specifically designed features for "measuring" and checking mathematical models.

5. VISUALIZING SENSOR DATA

In the previous section we discussed the evaluation of computer simulations, which were based on simple assumptions like the the behaviour of gas molecules. mathematical models (like Henry's Law) that are found by checking simulations are basically not more than logical deductions from the model the simulation is based upon. This section discusses the automatized evaluation of empirical data, gained by observing real experiments. The Raspberry Pi supports experimenting with all kinds of sensors (luminosity, temperature,ethanol, methane, carbon dioxide etc.).

Fig. 5 shows an NDIR (non-dispersive infrared) carbon dioxide sensor, connected via SPI (serial peripheral interface) to a Raspberry Pi. It is not very difficult to create a Python program that reads the sensor data and collects them in a list. The list of numbers can easily be plotted using the Python module MathPlotLib. A simple experiment with this sensor is the observation of gas diffusion. The NDIR-sensor is a tube with holes allowing gas molecules to enter and to leave. Inside the tube the CO_2-concentration is measured by detecting the absorbance of infrared radiation. When a person breathes against the sensor, the CO_2-concentration increases, because more CO_2 gets into the tube through the holes. After a few seconds it decreases again, because the gas molecules move (diffusion) and leave the tube through the holes.

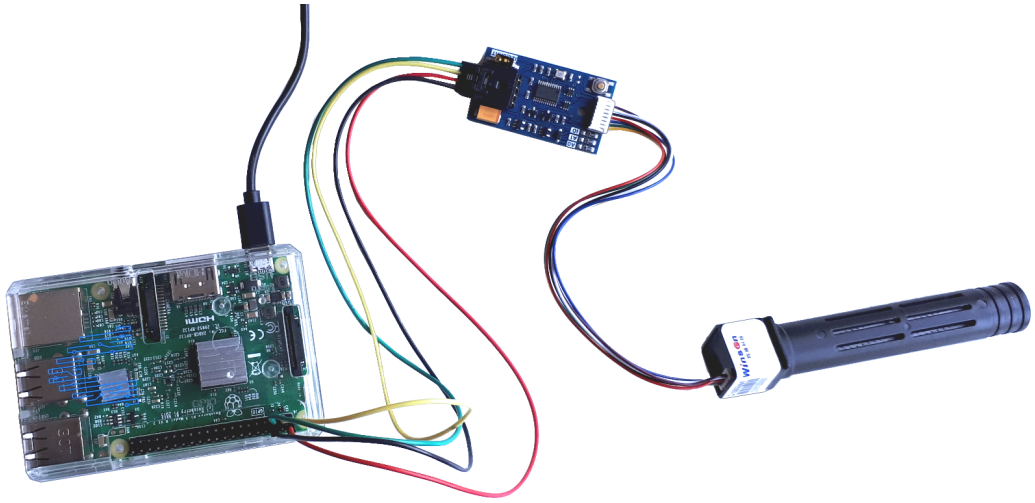

Figure 5. NDIR-CO_2-sensor connected to the GPIO of a Raspberry Pi

Fig. 6 shows the output of a Python program that semi-automatically conducts the diffusion experiment, collects the sensor data and creates a plot (using MatPLotLib). A further challenge is to formulate a mathematical model for the change of the concentration of CO_2 in the tube, write Python code to calculate a list of numbers, create a plot visualizing this list (using MatPlotLib) and compare it with the plot based on empirical data.

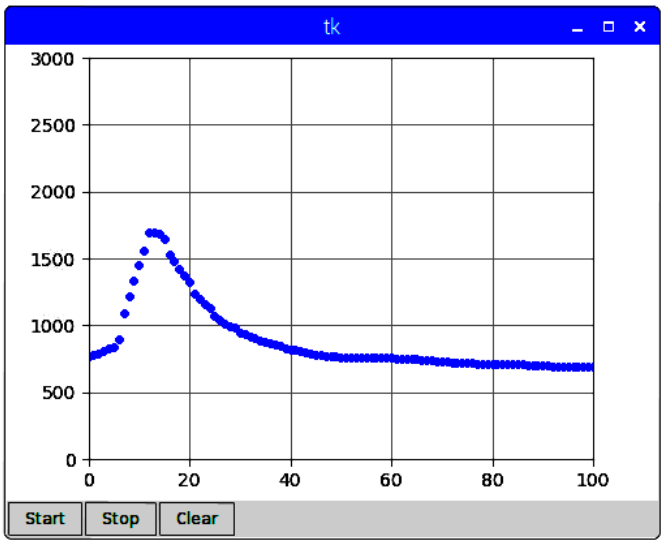

Figure 6. Plot visualizing the change of CO_2-concentration due to diffusion

6. CONCLUSION

Programming projects can be an enrichment of science classes. They offer many different ways to elaborate, check or even develop mathematical models of the physical world. This diversity makes it possible that even those students that are not primarily interested in math can find motivation to learn theoretical content. Since a programming project can be open and lead

to different products, learning is more individual. An open question is, to what extend students can develop programming competence on their own (without teacher's help) during projects in regular sciences classes. Physics, chemistry and biology teacher are not necessarily programming experts. One type of support are media (movies, tutorials, skill cards), specifically designed for science-related projects.

References

1. S. Papert, *Mindstorms: Children, computers, and powerful ideas*, Basic Books, New York, 1980.
2. M. Resnick, J. Maloney, A. M. Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for All," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.
3. V. Potkonjak, M. Gardner, V. Callaghan, P. Mattila, C. Guetl, V. M.Petrović, and K. Jovanović, "Virtual laboratories for education in science, technology, and engineering: A review," *Computers and Education*, vol. 95, pp. 309–327, 2016.
4. N. Tausch and M. von Wachtendonk, *Chemie 2000+ Qualifikationsphase*, Buchner Verlag, Bamberg, 2015.
5. L. Borghi, A. De Ambrosis, N. Lamberti, and P. Mascheretti, "A teaching–learning sequence on free fall motion," *Physics education*, vol. 40, no. 3, pp. 266–273, 2005.
6. J. M. Wing, "Computational Thinking," In *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

Received 12.04.2019, the final version — 16.05.2019.

Компьютерные инструменты в образовании, 2019

№ 2: 55–64

УДК: 004.942

http://cte.eltech.ru

doi:10.32603/2071-2340-2019-2-55-64

Математическое моделирование и программирование в естественнонаучном образовании

Вайгенд М.[1,2], доктор наук, mw@creative-informatics.de

[1]Хольцкампская общеобразовательная школа, Виттен, Германия
[2]Вестфальский университет Вильгельма, Мюнстер, Германия

Аннотация

Использование математических моделей для представления аспектов физической реальности является важной деятельностью в науке и научном образовании. В данной статье рассматриваются четыре подхода к использованию компьютерного программирования и математического моделирования в учебной деятельности:

1. Математические модели, найденные в учебнике, используются в качестве основы для компьютерных программ. Студенты, создавая полезные интерактивные программы на языке Python, рассчитывающие концентрации или значения pH, сталкиваются с такими же интеллектуальными проблемы, как при решении традиционных задач учебника.

2. Scratch-анимации, имитирующие моделирование физических или химических систем, могут быть специально разработаны для проверки достоверности заданных математических моделей.

3. Задача, связанная с компьютером, заключается в разработке моделирования (например, диффузии газа в замкнутой системе с двумя фазами), что может быть основой для открытия математической модели (например, закона Генри) или элемента математической модели.

4. Используя сенсорную технологию и Raspberry Pi, студенты создают компьютерную программу, которая автоматически визуализирует наблюдаемое поведение системы (например, изменение концентрации газа), чтобы в дальнейшем разработать математическую модель.

Ключевые слова: *наука образование, Python, Scratch, программирование.*

Цитирование: Вайгенд М. Математическое моделирование и программирование в естественнонаучном образовании // Компьютерные инструменты в образовании. 2019. № 2. С. 55–64. doi: 10.32603/2071-2340-2019-2-55–64

Поступила в редакцию 12.04.2019, окончательный вариант — 16.05.2019.