Michael Kallweit, Jonas Lache (2021): Playing with interactions in STACK exercises.
In: Contributions to the International Meeting of the STACK Community 2021. TTK University of
Applied Sciences: Tallinn, Estonia. DOI: 10.5281/zenodo.4916013

Playing with interactions in STACK exercises

Michael Kallweit[1], Jonas Lache[2]

Abstract: By using question behaviors with multiple tries, we bring more interactivity into STACK tasks. On the front end side, besides using JSXGraph as a tool to try out and explore, we also work with additional small JavaScript snippets. On the back end side, we make heavy use of the possibilities to work with strings generating HTML code. As a result, we get new variants of using STACK as a learning environment that positively influences student understanding.

Keywords: Interactivity; JSXGraph; Strings; JavaScript; STACK

1 Introduction

The assessment system STACK has proven to be an invaluable tool for the digitization of teaching at universities. It has been used beneficially in various scenarios at the Ruhr-University Bochum for many years. Many tasks have been developed, and plenty of them make best use of the system's capabilities. But there are still things that can make STACK even more useful. In collaboration with lecturers, we design new desirable and useful features and develop first technical implementations. We do this by creating an additional layer based on JavaScript or by using the underlying computer algebra system in a clever way. Our main goal is to implement ideas that have proven useful as extensions to STACK. We are building on the preliminary work and ideas of [KG19] and [Al20].

2 Generating dynamic content

In this chapter, we introduce a new approach to using the computer algebra system (CAS) Maxima to create STACK questions. When authoring a STACK task, the CAS is typically used for different purposes, for example generating question variables and evaluating the students' input. But besides the possibility to deal with mathematical expressions, Maxima can also generate and work with strings, meaning sequences of alphanumerical or numerical characters that are not treated as mathematical expressions. In combination

[1] Ruhr-Universität Bochum, Fakultät für Mathematik, Universitätsstr. 150 , 44780 Bochum, Germany
michael.kallweit@rub.de
[2] Ruhr-Universität Bochum, Fakultät für Mathematik, Universitätsstr. 150 , 44780 Bochum, Germany
jonas.lache@rub.de

with Maxima's functions to program conditional statements and loops, as well as the functions for randomization offered by STACK, we let Maxima create HTML or JavaScript code. This makes it possible to create a new kind of dynamic content within STACK questions, providing new opportunities for creating digital tasks that have not been discussed before. We demonstrate two examples below: an improvement of the specific feedback for multiple-choice questions (input type "Checkbox") and a way to generate a dynamic number of input fields including the opportunity to let the students decide how many input fields they need.

2.1 Feedback improvement for multiple-choice

The benefits of STACK include the possibility to provide students with individual, specific feedback and a large variety of possible input types to implement opened and closed questions. Regarding multiple-choice questions, Mason and Bruning (2001) suggest that, feedback with individual item verification (*knowledge of correct response*) can be beneficial, especially for students at a lower level of achievement [MB01]. With this in mind, the feedback options for multiple-choice questions offered by the standard features of STACK must be considered as limited, as we see from the following example where a polynomial with the equation $f(x) = x^2 - 4x + 3$ ist given. The students are asked to select all roots of the polynomial by checking the corresponding checkboxes, whereby the options -1, 1, 2 and 3 are available. According to Mason and Bruning, the students should be provided with an individual item verification after submitting the task. This could be done by providing verification symbols. However, when using the set of the selected answers as the student answer (`SAns: setify(ans1)`), the set of the correct roots as the teacher answer (`TAns: set(1,2)`) and "Symbol only" as the PRT feedback style, STACK only gives a verification for the whole task and not for each item (Fig. 1, on the left). This is not the intended verification. In most cases, using the standard Moodle question type for multiple-choice tasks is not a proper solution either, as it is impossible to use the CAS (i.e., for randomization) there.

To address this issue and get verification symbols for each item, we found a new way to give feedback for multiple-choice questions implemented with STACK. First of all, we create a PRT node for each item and let STACK compare whether the corresponding root *has been checked* (`SAns: member(<item>, setify(ans1))`) and whether it *was to be checked* (`TAns: member(<item>, set(1,3))`). In the feedback texts, we generate HTML input fields with the type "checkbox" followed by a verification symbol. To show the students if they checked the item or not, each of these checkboxes is checked or unchecked depending on whether the item was selected before. This works by Maxima adding a string to the HTML code in each node (i.e., for each item). Each string has a value depending on whether the corresponding item was checked by the students, for example, for the first item:

```
checked1 : if member(list[1],[1],ans1) then "checked" else "";
```

The question variable `list` is the "model answer" for the STACK "Checkbox" question. The HTML code then looks as follows:

```
<input type="checkbox" disabled {@checked1@}> \({@list[1][1]@}\) </input>
```

This leads to an appropriate result. However, it is also possible to hide the input field when the feedback is shown (Fig. 1, on the right). This can be achieved by placing the placeholders for the input and verification inside an HTML div section:

```
<div id="ans_input"><p>[[input:ans1]] [[validation:ans1]]</p></div>
```

In the PRT, a JavaScript function is called that hides the div section as soon as the page has been fully loaded:

```
window.onload=function(){
document.getElementById('ans_input').style.display="none";
};
```

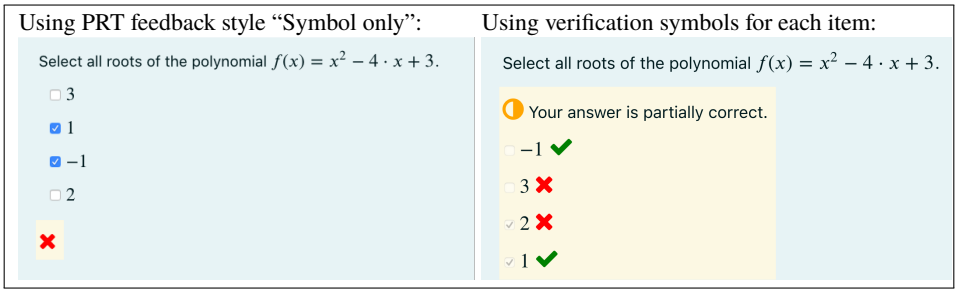

Fig. 1: Feedback for multiple-choice questions.

2.2 Dynamic number of input fields

With a similar idea, we can generate a variable number of input fields for each attempt and still be able to evaluate them with STACK. This comes in handy when the number of answers to be given actually depends on a random object. The first idea is to generate proper HTML code in a Maxima string for many as needed input fields. In this example, we create four input fields with individual ids:

```
inputs:makelist(sconcat("<input id='inp",i,"'><br>"),i,1,4);
```

Then we just add this string variable to the question text at the required position.

Our second idea takes this a step further and allows the user to generate as many input fields as needed at the task runtime. We provide a button which creates inputs fields on the fly, see example in Fig. 2.

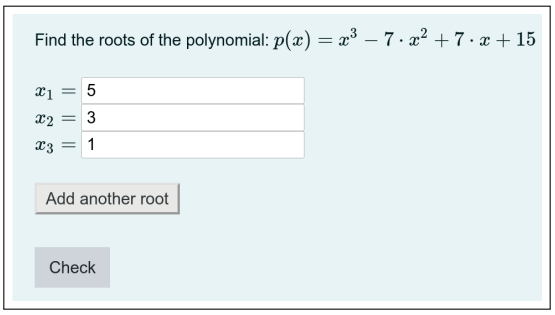

Fig. 2: Generation of input fields by the user.

We achieve this by dynamically adding new DOM elements (via the JavaScript function `document.createElement`) to the HTML DOM. In the background, the values of the generated input fields are converted to a list, which will be written back and forth to a hidden STACK input field. This list can then be evaluated normally in a response tree. Although the order of the entries is preserved, it is usually useful to work with sets.

3 Saving the current state

Hidden input fields can also be used to store information about the question attempt itself. For example, in a task where the user has the possibility to access hints (Fig. 3), it could be tracked if and which hints were really used. This information can then be used to adjust the generated feedback.

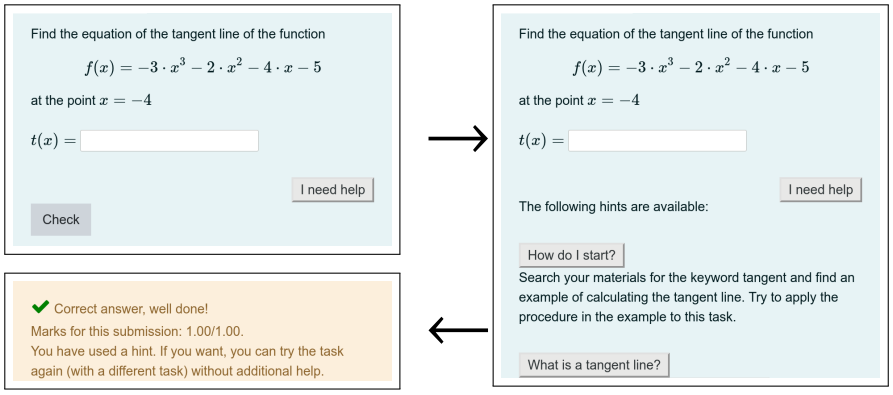

Fig. 3: A question with provided hints.

Our hope is that this approach will allow to create even more appropriate feedback that will be received even better by students.

4 Incorporating external JavaScript libraries

Nowadays, almost all interactions happening in web browsers work with the programming language JavaScript in the background. In the previous sections of this paper, we already gave several examples on how to use JavaScript snippets to bring more interaction into STACK tasks. But in addition to writing one's own JavaScript code, it can also be sensible to make use of external JavaScript libraries.

4.1 Using JSXGraph in new ways

The University of Bayreuth in Germany provides the powerful JavaScript library JSXGraph. It supports plotting of mathematical objects in the web browser. Besides static graphics, it is also possible to generate interactive dynamic graphics where the user can change the objects. Since there is an application programming interface (API) from JSXGraph to STACK, the use of JSXGraph in STACK questions for visualization of the task or as a graphic input is frequently used and has been discussed before (e.g., [KBV19]). However, the way we use JSXGraph is different: First, we use JSXGraph as an interactive application in the feedback. Second, we use it as a digital tool in the question text.

For some tasks, in which an expression with a specific property is to be entered, it can be sensible for students to get a visual representation of their answer in the feedback to check if they were correct (i.e., if the input fulfills the requested property). One example of such a task is the following:

> *Please specify two different points P and Q so that the line through P and Q has the equation* $y = \frac{x}{2} + 2$

Here, students could be provided with **graphics in the feedback** that show the points they specified. They then could see if the resulting line equals the line with the given equation. Thinking ahead a little, it is also possible to provide a *dynamic* image in which the students can change the elements to correct their initial input. This would combine the graphic input of STACK using JSXGraph with the specific feedback. Furthermore, using a visual way to solve the problem could bring benefits to the students, as it can provide them with another approach to the mathematical concept (cf. Krämer et al. (2012) who introduced a classification of different solutions for modeling exercises [KSB12]). Fig. 4 shows how this can look. The students are provided with an interactive graphic and are prompted to move the points to correct their initial answer. When they think that the points are in the correct position, they can click on a button. This calls a function that checks if the line is now equal to the line from the question text. If so, the background of the graphic changes from orange

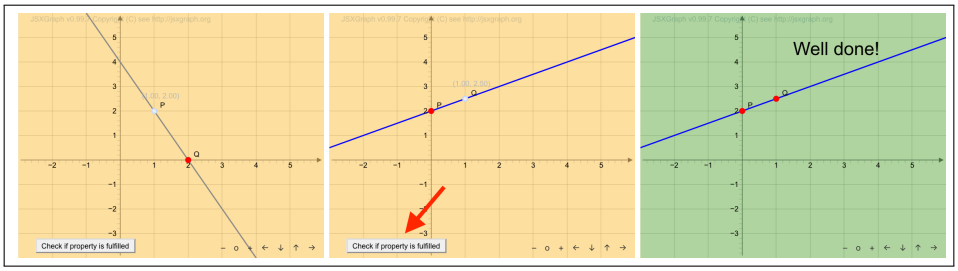

Fig. 4: Students are asked to move the points given in the graphic which appears in the feedback.

to green, and the text “Well done!” appears. If the answer is still wrong, the students can try again. When the students have come to a correct solution, using the interactive tool, it could be sensible to let them reflect on their initial solution and incite them to think about a way to solve the task using algebraic methods. For example, students could be asked as follows: “Now please go back to the initial task. How can you mathematically find two points that lead to the line equation $y = \frac{x}{2} + 2$?”.

Our second approach to using JSXGraph in a new way is to use it as a **digital tool in the question text**. As suggested by Winter (2016), trying things out and exploring can bring benefits for the learning of mathematics, because students can build conceptions (active discovery learning) [Wi16]. This implies that it could be helpful for students to try out their ideas before submitting a final answer. In our example, the students are supposed to specify a function that is continuous but not differentiable in $x_0 = 3$. With a plotting tool in the question text, students can approach a correct solution step by step, for example by first finding a line with a root at $x_0 = 3$ and then applying the absolute value to this function to make it not differentiable at this position. Such a digital tool can be implemented using JSXGraph and then embedded into a STACK question (Fig. 5).

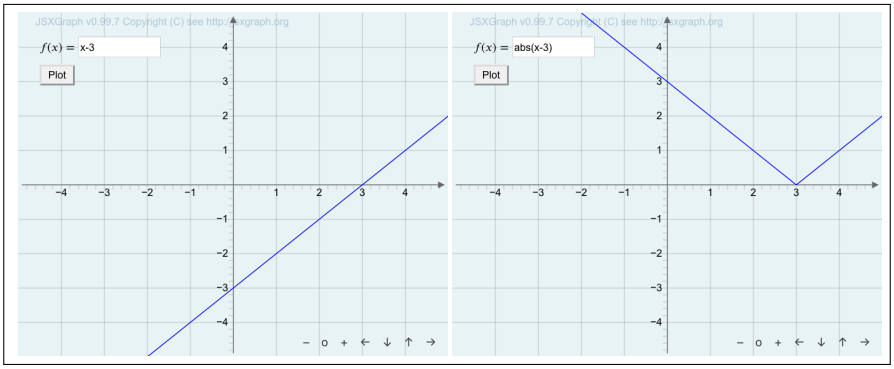

Fig. 5: Before submitting their answer, students can try out their ideas by plotting functions.

The particularity here is that there is no evaluation by STACK as long as the students click on the button within the graphic. Only when they click on “Check” or “Finish attempt”, the input will be evaluated. Until that, students can plot new functions as often as they want to. To avoid the situation that the students would have to type in their final answer twice, the term from the input field in the graphic is automatically entered into the STACK input field. This happens every time the “Plot” button is clicked and it works by using the JavaScript querySelector function.

4.2 Integrating chats

Of course, there are many JavaScript libraries other than JSXGraph that can be used within a STACK question to make it more interactive. We believe that, despite all the great technology, the best interaction is still with people. We want students to work together and learn together, even in the time of a global pandemic. One obvious approach to enable cooperation is to integrate a chat right next to the task. This enables communication and mutual assistance. We have built a task that makes cooperation necessary, see Fig. 6.

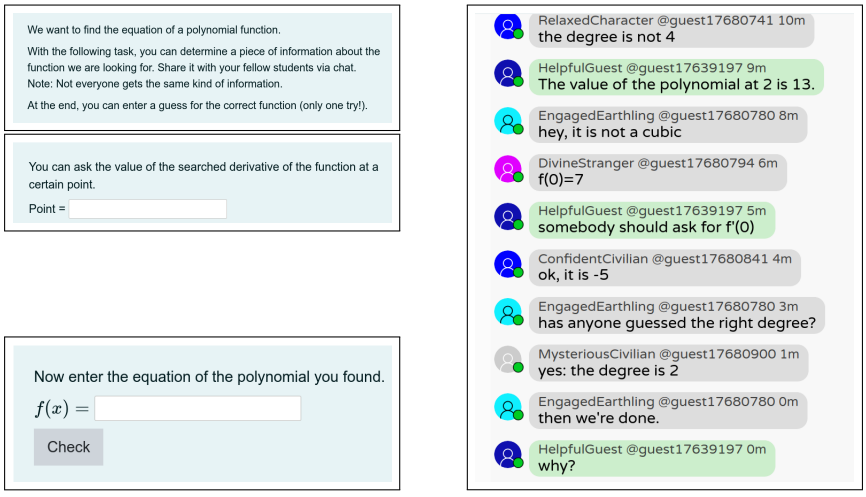

Fig. 6: A collaborative exercise with global chat.

In this exercise, the students must find a secret polynomial function. Everybody can get one piece of information, but also different kinds of it (for example getting the value at a chosen point or guessing the degree). Which type of information a student can access is chosen at random and the user just uses it once. It needs cooperation and agreement to find the correct solution.

5 Future work

We are always on the lookout for new interesting ideas that will enhance the usage of STACK. In this work, we have shown a number of concepts on how to add interactivity to STACK exercises. Related to our approaches presented here, we are pursuing other techniques as well. Here we show two examples in a nutshell.

First, we think about further developing our idea to include interactive graphics in the feedback to let students change their answers graphically and provide them with a new approach to the task. Here, we plan to give the graphic input back to STACK and let Maxima evaluate it. Using another PRT, it would be possible to provide students with more detailed feedback than it is possible when only using JSXGraph instead. Although binding the objects within JSXGraph in the PRT with a STACK input field is not possible using the standard binding logic offered by STACK, it could work using the JavaScript querySelector function.

Second, we presented an example at the STACK community meeting 2021 that did not make it into this text which lets the students decide what input type they want to use for a task. This works by using a lot of JavaScript, HTML, and hidden STACK input fields.

Ready-to-use sample tasks of the concepts we presented in this paper can be found on our website → `https://www.rub.de/hdm/stack.en.shtml`.

References

[Al20] Altieri, Mike; Horst, Jörg; Kallweit, Michael; Landenfeld, Karin; Persike, Malte: Multi-step procedures in STACK tasks with adaptive flow control. In: Contributions to the 3rd International STACK conference 2020. Zenodo, July 2020.

[KBV19] Klischat, Cosima; Becker, Peter; Vasko, Mikko: STACK is more than Maths – Development of Online - Problems for Mechanics and Electrotechnics. In: Contributions to the 1st International STACK conference 2018. Friedrich-Alexander-Universität Erlangen-Nürnberg, Fürth, Germany, 2019.

[KG19] Kallweit, Michael; Glasmachers, Eva: Adaptive Selbstlernaufgaben mit STACK. In: Contributions to the 1st International STACK conference 2018. Friedrich-Alexander-Universität Erlangen-Nürnberg, Fürth, Germany, February 2019.

[KSB12] Krämer, Jana; Schukajlow, Stanislaw; Blum, Werner: Bearbeitungsmuster von Schülern bei der Lösung von Modellierungsaufgaben zum Inhaltsbereich Lineare Funktionen. mathematica didactica, 35:50–72, 2012.

[MB01] Mason, Beryl Jean; Bruning, Roger H.: Providing feedback in computer-based instruction: What the research tells us. Technical Report January 2001, Center for Instructional Innovation, University of Nebraska-Lincoln, 2001.

[Wi16] Winter, Heinrich Winand: Entdeckendes Lernen im Mathematikunterricht. Springer Science and Business Media Deutschland GmbH, Wiesbaden, 3 edition, 2016.