Chris Sangwin

The DragMath equation editor

Chris Sangwin
Maths Stats and OR Network
University of Birmingham
C.J.Sangwin@bham.ac.uk

Background

I joined the MSOR Network in September 2000 having just finished a PhD in Mathematics at the University of Bath. Until September 2011 I devoted about half my time to the Network and undertook a number of learning and teaching related projects. These include a continuous development of automatic assessment using computer algebra. The work on automatic assessment contains quite a variety of aspects, and this articles concentrates on one of these.

One of the challenges of automatic assessment is how to enter a mathematical expression into a machine. This is a fundamental computer interface problem. There are a number of options, perhaps the most basic being a typed linear syntax, so that $\frac{1}{1+3x^2}$ might be entered as `1/(1+3*x^2)`. Another common approach is to build the components by dragging them to assemble the expression in a visual way. A third alternative is a hybrid, where the two dimensional expression builds up as you type. This combines the simplicity and speed of typing with the immediate visual appeal of two-dimensional traditional notation. For example, the NUMBAS system uses this approach, **http://www.ncl.ac.uk/maths/numbas/**

I wanted to provide a drag-and-drop interface to the computer aided assessment system STACK, [8, 7]. However, I could not find a suitable freely available, i.e. open source, equation editor. To address this need, during 2006-2007 I ran a Computer Science Final Year Project at the University of Birmingham. The goal of the project was "***to design and create an application that enables a user to easily enter mathematics into a computer. The interface will allow users to pick desired mathematical components by clicking buttons or dragging and dropping onto a workspace in order to generate mathematical expressions***." A potential secondary goal of this project was to further develop the application into a fully-featured and useful equation editor tool which could be incorporated into the STACK computer aided assessment system.

One of the difficulties of running this project is that it has been done before. Another is that it looks trivial. However, I am unconcerned about the first of these difficulties: ***mathematics education is the art of reinventing the wheel***. With this point of view, it makes an interesting student project. The second is more problematic. One of the significant lessons I have learned from automatic assessment is that to automate a process you have to understand it intimately. Mathematical notation looks simple, indeed it has evolved precisely that way. However, in the next section I want to provide some examples of why mathematical notation, and hence the design of any mathematical interface, is an interesting and challenging design problem.

Why notation is interesting

In this section I would at least like to illustrate why mathematical notation is non-trivial. Mathematical notation and its meaning is something we perhaps take for granted.

Examples of the power of a well contrived notation to condense into small space, a meaning which would in ordinary language require several lines or even pages, can hardly have escaped the notice of most of my readers. [1, pg 331]

Mathematical notation has evolved over a long period of time and takes advantage of a rich set of special symbols, together with their relative size and position on a two dimensional page. The two most basic ways symbols are combined are

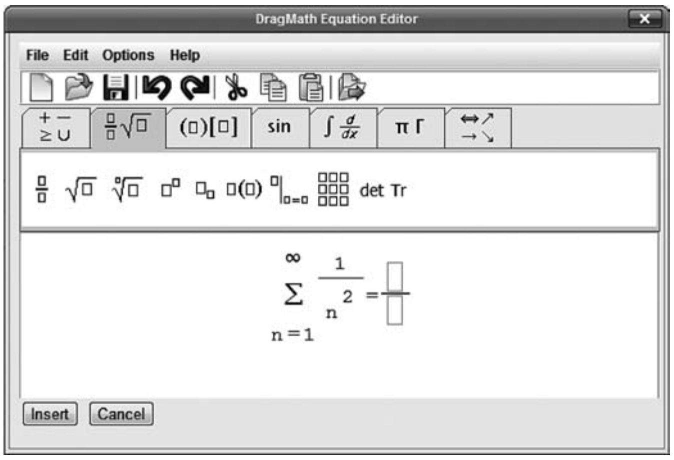
1. *juxtaposition*, by placing them next to each other e.g. $2x$,
2. *grouping*, by combining many symbols and treating them as a single unit.

The traditional contemporary use, of both of these, result in some significant ambiguities. For example, juxtaposition sometimes means multiplication, as in $2x$, but it sometimes means addition, as in $2\frac{1}{2}$ and at other times it could be function application, as in $\sin x$. Parentheses are often used for grouping, as in $2(x+1)$, for function application $\cos(n\pi)$ (which is arguably also grouping), for denoting real intervals, e.g. $(-1, 1)$ and other specialist uses. Relative size and position, e.g. superscript, provides ambiguity between powers and other function application: compare $\sin^2(x)$ which is often interpreted as the square of $\sin(x)$ (not $\sin(\sin(x))$) whereas $\sin^{-1}(x)$ is the inverse function (not the reciprocal). There are also cultural differences in notation, and interesting historical developments, see [5] and [2].

If you have serious doubts about how powerful notation can be, try changing something. For example, in [4] Brown proposed a change to the notation for logarithms, motivated by computer notation for exponentiation a^b as $a\uparrow b$, or perhaps more commonly now `a^b`. She suggested that the logarithm of b to the base a, sometimes written as $\log_a(b)$, could be written $a\downarrow b$. We generally don't have a keyboard symbol ↓, so instead we could type `a_b`, and display this as a_b. Then the natural logarithm becomes simply e_x. In this new notation, which of the following are correct laws of logarithms?

$a^{a_x} = a_{a^x} = x,$
$a_{b\times c} = a_b + a_c,$
$a_{b^c} = (a_b) \times c,$
$(a_b) \times (b_a) = 1,$
$(a_b) = (a_c) \times (c_b).$

'Correct', at least in the elementary sense. Normally we gloss over issues of branch cuts and complex numbers when introducing logarithms to students. In what sense are algebraic identities like this 'correct'? One of the goals of DragMath was to let the user build up expressions, but internally to try to capture the *intended meaning* in an unobtrusive way. In parallel to the DragMath project, this interest in notation motivated us to investigate linear input syntax for STACK and CAA in general, e.g. [9] where more of these issues are considered.



$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\square}{\square}$$

Fig 1 – The DragMath equation editor

What is DragMath?

The project ran in 2006-2007, and I was fortunate in attracting Alex Billingsley as the project student, see [3]. As is appropriate for a project, exemplars of each 'type' of operator were added. E.g. Alex added some trigonometrical functions, but not the full list. Subsequently I paid Alex from my research funds to add further features, to fill out the functions list, and to make changes which I wanted, but which were not relevant to Alex's project. The final result was DragMath. See **www.dragmath.bham.ac.uk**. DragMath is a 'drag and drop' equation editor. It is a Java applet which can run within a web browser. The editor lets users build up mathematical expressions in a traditional two dimensional way, and then output the results in a correctly formed syntax.

DragMath is based on the idea of templates for each operator or function. Templates can be inserted which consist of the traditional notation and layout for that particular operator/function and also blank boxes. The number of boxes depends on the number of arguments the operator/function takes. An example is shown in the right of Fig 1. Here the two-dimensional fraction operator has boxes on the top and bottom which do not yet have any contents. On the left, the summation operator has four arguments supplied, the variable name, the upper and lower limits and the summand.

To insert a template from the toolbar you can *point and click* or *drag and drop* onto the workspace area, or an existing text box. Notice that templates are grouped in tabs, which is a common form of user interface. Ultimately, DragMath incorporated a simple parser (using 3rd party code) so that typed expressions, e.g. `1+x^2` would also be correctly interpreted. This gave the interface flexibility and enabled more experienced users to type when this is more efficient. Basic editing options are supplied to edit the expression, including select, copy, paste, delete, redo, undo etc. Deciding on how 'select' should work is also a non-trivial task. Further comments on this problem are given by [6].

A key design feature of DragMath is the ability to configure the applet without having to recompile the code. This extends in three major ways.

- There are various *parameters* that can be set inside the `<APPLET>` tag, to change particular options within the applet.
- The user can write their own *language versions*.
- The user can write their own *output format*.

As might be expected, there are a number of applet parameters, including loading the applet with an initial expression. This is crucial if we return to an expression to edit it later. We can also show and hide menus, including various tabs.

It is possible to convert the expression to almost any format desired by creating an XML file with the correct syntax data in it. We will discuss this in more detail below. Note, that users can adapt the format of the output without recompiling the applet. DragMath is supplied with seven output format files including LaTeX, MathML, Maxima and Maple syntax.

Adding a new mathematical operator or function requires developer access. Designing a platform in which this was abstracted into an external configuration file was unnecessarily complicated for our purposes. However, we have received a number of requests to add functionality. In particular, the following recent request from chemists is not unique.

I find the tool super user friendly to create mathML but there is one simple thing that is really holding us up.

We can't figure out how to write something like H^2O.

What is the sequence that we have to do to enter a letter, a superscript number and another letter?

In DragMath we could certainly add functionality to do this, but what does H^2O actually mean? Perhaps we need some kind of chemical bond operator, because these juxtaposed symbols certainly have some meaning. DragMath seeks to encode the meaning internally. Once this is done we can output a representation of the expression in many formats, not just a presentational form such as H^2O.

Design of DragMath

Internally, DragMath uses an *abstract expression tree* (AET) to represent mathematical expressions. Operators are placed on nodes, and the operands are on the leaves. The operands could also be an AET or an atomic expression, such as a number or abstract symbol for a variable. DragMath needs to deal both with well-formed and ill-formed trees. As the user builds up an expression in stages, as in Fig 1, the expression is usually incomplete.

Each operator belongs to a particular group, e.g. unary operators take one argument, as in sin(x) or $n!$. Binary operators have two arguments, although associative binary operators are normally represented as so-called n-ary operators. As a specific example addition is strictly speaking a binary operator, although we traditionally write expressions such as $a + b + c$. Rather than adhere to a strict binary operator, which would force the user to choose between $(a + b)+ c$ and $a +(b + c)$ DragMath quietly flattens such expressions to give $a + b + c$. If the user really wants $a +(b + c)$ then the last two terms can be explicitly grouped together. Some CAS, such as Maxima, have an internal representation which permit such n-ary operators, using a syntax such as

```
"+"(a,b,c)
```

Here, the function + can accept an arbitrary number of arguments.

One of the key design features is a mapping of the abstract expression tree onto a required output format. Take the expression $\int_a^b f(x)\mathrm{d}x$. In LaTeX this might be typeset using

```
$\int_a^b f(x) \mathrm{d}x$
```

In Maple

```
int(f(x),x=a..b);
```

and in Maxima

```
int(f(x),x,a,b);
```

The MathML version is too horrible to reproduce here, but the principle is clear. In some cases, the orders of arguments changes. DragMath loads an XML file which encapsulates the rules for the intended output format. It then applies these rules to the expression. Hence, if a user wishes to modify the rules they can do so without changing the applet itself. Indeed, it is possible to create whole new output formats.

One of the difficulties Alex had in presenting his work was the simplicity of his final design and the applet he implemented masked the sophisticated data structure and the flexibility of the output mechanism.

Uptake and use

When making choices about what to spend time doing, it is not clear what will ultimately be most valuable or will be used by others. DragMath is a case in point. Originally it was designed to provide a drag-and-drop editor for the STACK computer aided assessment system, but it was always our intention that this should have a separate project identity. Since its release in the summer of 2007, DragMath has become very widely used in a variety of other projects. It is likely that we do not know about all of these.

There are a number of reasons why DragMath has been widely used. Firstly there was a need for this tool, which was not satisfied by any other software. Alex's code has proved to be very reliable, and made the job of equation editor appear simple. Part of the design was to ensure it was possible to integrate DragMath into STACK, but we made sure this was general and could be used elsewhere.

Perhaps the most important reason why DragMath has been widely used is that we made it *easy for others to adapt DragMath*. In particular, it is not necessary to recompile the applet to change the language. Others have contributed translations of DragMath into: Catalan, Czech, Dutch, Finnish, French, German, Italian, Norwegian, Persian, Polish, Portuguese (Brazilian), Russian, Turkish, Spanish and Swedish. There may be other translations of which we are not aware.

The largest source of users however make use of DragMath through the TinyMCE editor. This is a WYSIWYG editor control for web browsers that enables the user to edit HTML content in a more user friendly way. The editor control is very flexible and is built for integration purposes. DragMath has been plugged into this editor, and returns the LaTeX displayed form of an expression. TinyMCE is the default editor for the Moodle VLE. As of Moodle 2.x DragMath is now distributed as part of the Moodle core. To use DragMath, you are required to activate the TeX filter and ensure that you have Java installed. So, if you have Moodle version 2.x, you automatically have DragMath already. There are many other systems using TinyMCE, and it is likely that many of those which need support for mathematics will also use DragMath through TinyMCE.

DragMath has been released under the GNU General Public Licence. The source-code has been downloaded 8,287 times since Aug 2007 and since July 2007 the compiled applet has been downloaded 46,508 times from the SourceForge site. Of course, most users will access a compiled version of the applet from another website making the extent of DragMath use hard to establish accurately. It is very difficult to establish how many different projects make use of it, although the numbers using DragMath through Moodle alone are now substantial. The site `http://moodle.org/stats/` contains statistics of Moodle usage.

Conclusions

It was clear when I suggested a drag-and-drop equation editor as a final year computer science project that it was interesting both intellectually and as a computer science project, even if it looked 'trivial'. I also had a clear need for such software in practice, and did not have the time to write the code myself; instead I contributed to the design discussions. What started as a pragmatic desire for an editor, developed through a student project into both useful software in its own right, and in parallel for me as a modest research interest, see [9].

One reason for the take-up is the ease with which others can integrate DragMath into other software and modify parts of the behaviour, e.g. the language and the output format, without having to recompile the applet.

I am very grateful to Alex for his work on DragMath and his continuing interest in supporting this project in his own time. I would also like to acknowledge the help of colleagues who have integrated DragMath into Moodle, and translated the applet into other languages.

References

1. C. Babbage. On the influence of signs in mathematical reasoning. *Transactions of the Cambridge Philosophical Society*, II:325–377, 1827.
2. C. Babbage. On notations. *Edinburgh Encyclopaedia*, 15:394–9, 1830.
3. A. Billingsley. Input of elementary mathematics. BSc mathematics and computer science, University of Birmingham, 2007.
4. M. Brown. Some thoughts on the use of computer symbols in mathematics. *The Mathematical Gazette*, 58(404):78–79, June 1974.
5. F. Cajori. *A history of mathematical notations*. Open Court, 1928.
6. J. F. Nicaud, D. Bouhineau, and H. Chaachoua. Mixing microworlds and CAS features in building computer systems that help students learn algebra. *International Journal of Computers for Mathematical Learning*, 9(2):169–211, 2004.
7. C. J. Sangwin. Computer Aided Assessment of Mathematics Using STACK. *In Proceedings of ICME 12*, 2012.
8. C. J. Sangwin and M. J. Grove. STACK: addressing the needs of the "neglected learners". In Proceedings of the First WebALT *Conference and Exhibition January 5-6, Technical University of Eindhoven, Netherlands*, pages 81–95. Oy WebALT Inc, University of Helsinki, ISBN 952-99666-0-1, 2006.
9. C. J. Sangwin and P. Ramsden. Linear syntax for communicating elementary mathematics. *Journal of Symbolic Computation*, 42(9):902–934, 2007.