Interactive Java Applet for Equation Derivations

Kenneth S. Manning, Ph.D. and Luke B. Bellandi
Rensselaer Polytechnic Institute

Abstract

The Equation Activity applet, developed by Project Links at Rensselaer, is an engaging and interactive tool that allows instructors to guide students through the steps in deriving a particular equation. Project Links, an NSF-supported project at Rensselaer, is a cooperative effort by faculty from several departments, schools, and institutions to develop materials linking mathematical topics with their applications in engineering and science. The primary product of this effort is a set of interactive, web-based learning modules that rely heavily on hypertext, animations, and interactive Java applets.

Through the Equation Activity, the instructor gives students enough information to derive a particular equation. The student's response is then evaluated by use of a semi-intelligent algorithm that recognizes association, commutativity, distribution, implicit multiplication, legal re-ordering of terms (flipping about the equals signs), and unlimited legal use of parentheses.

The instructor specifies the particular terms, variables, operators, and constants for use by the student on a customizable keypad. This keypad can vary as the derivation steps progress, reflecting new information gained. The path to the desired equation is broken into several steps, allowing the student to progressively work toward the final equation. For each step in the activity, the instructor, using text and equations, specifies the steps to take to obtain the next intermediary equation. Hints can be supplied to the student who enters a wrong or inappropriate equation. Once the student has produced the proper equation for a step, that equation is saved and can be reviewed by the students as they work on subsequent steps. This process iterates until the final equation has been reached.

The applet is easily modified and added to existing courseware.

Background

Project Links[1] is a five-year, NSF supported undertaking to develop web-based interactive modules that integrate mathematical concepts with contemporary topics in science and engineering. The project is based at Rensselaer Polytechnic Institute, with collaboration from the University of Delaware, Virginia Polytechnic Institute, Hudson Valley Community College, and Siena College.

In the development of our learning modules we view interactivity and the continuing engagement of the student in the discovery process as paramount to our success. Two of our four formal main objectives[2] expressly state interactivity must be at the forefront of all of our work. In part, due to our ability to achieve and maintain this interactivity, we were awarded the ***Premier Award for Excellence in Engineering Education Courseware Award 2000*** by NEEDS: The National Engineering Education Delivery System[3].

One distinct area in which such true interactivity was highly desirable, but elusive in practice, was when guiding students through the derivation of equations. Limited interactivity is attainable by stepping through a question-and-answer process, where students are given much of the information and then asked about certain parts of it before proceeding. But we wanted the students to make the attempt at a step in the derivation and have that checked before the student could proceed. The Equation Activity we discuss here is the outgrowth of this desire.

Previous attempts at this type of activity involved either of two common assessment methods. The program could be pre-coded with as many possible correct answers as the developers could anticipate, and the students' answer was checked against this list for a match. Otherwise the students' equation and the correct equation could each be numerically evaluated, and those values compared to within certain set tolerances. The latter is called the *zero equivalence* problem[4].

The mission of Project Links is to develop educational materials that link mathematical topics with applications in engineering and science. We needed an activity in which module users could derive a complex equation through a series of guided steps.

In the Equation Activity, the student is given a modified calculator interface, as shown in Figure 1. In addition to the standard numeric buttons and arithmetic operator keys, the interface has buttons that make any variables available. These variables are specific to the equation the student is deriving, and are specified by the activity developer. The student is also given a text window called the "requirement window", shown as (1) in Figure 1, where the professor specifies the current equation to be built by the student. The student enters her answer in the "build window" (2) by using any buttons from the "build console" (3). Correct answers are available for reference in the "completed equations window" (4).

The Equation Activity can be seen in use at http://links.math.rpi.edu/devmodules/mechanicalosc/springmass/, choose "Math Model", then "Derivation Activity" near the bottom of the page, once it has loaded.

As a simple example, consider an activity whose goal is to derive the equation of motion for a given set-up as shown in Figure 2. Any necessary background and discussion can precede the Activity to prepare the student. The requirement window might instruct the student to "*State the general form of Newton's Second Law*" to which the student would (hopefully) respond $\sum \vec{F} = m\vec{a}$. The appropriate buttons for each of the five terms would be made available for this step by the professor. When the student has correctly entered the answer, the interface changes to one with a new question and buttons necessary for answering that question.

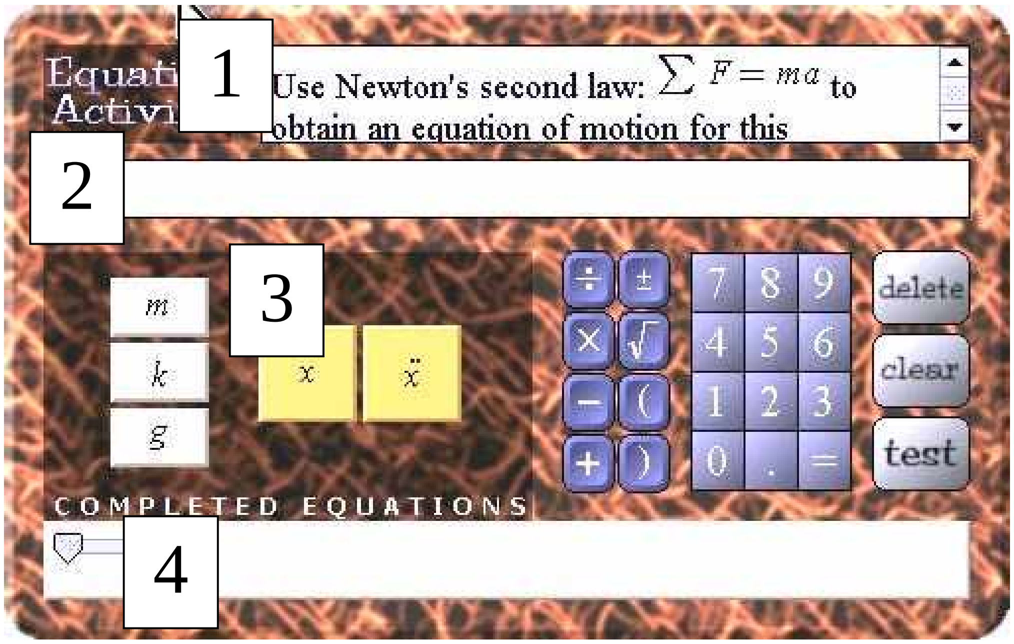

Figure 1. The Equation Activity Interface.

The Activity might now ask "*Apply Newton's Second Law in the x-direction*." The student might respond with $F - f = m\ddot{x}$ which would be acceptable. Other forms that are algebraically equivalent would also be considered correct. Once the student's answer has been judged as correct, the Activity changes yet again for the third (if desired) step in the process.

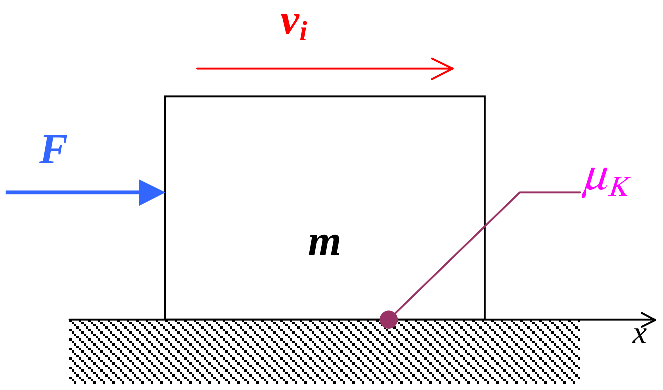

Figure 2. A simple example to illustrate the use of the Equation Activity.

Implementation

Initial implementation of the Equation Activity was done using HTML and JavaScript. As a prototype, it enabled us to review the design and make rapid changes. However, as a final product it was impractical. Using this method, new activities had to be custom built, in each case requiring a few dozen specialized graphics as well as a redefined interface. Also, this implementation could not handle basic rules of arithmetic such as associativity, commutativity that modify the arrangement of an equation, but not its value.

The final implementation was to incorporate all elements of the prototype with four additional or corrective changes.

First, the activity must provide a more robust comparison engine for the equation. Surely it is unfair to tell a student she is wrong for entering the answer $m\vec{a} = \vec{F}$ simply because the answer in the key is stored as $\vec{F} = m\vec{a}$. The system should be able to take into account principles of commutativity, associativity, distribution, appropriate arbitrary use of parentheses, arbitrary term placement about the equals sign, and implicit multiplication (adjacent terms). The implementation of this engine is discussed in the following section “Algorithm”.

Second, the activity development process should be streamlined. It should be as simple and consume as little time as possible for professors to create a new activity. The model we used is the video-game system model of “console” and “cartridge”. Project Links created the activity console incorporating as many standardized features as possible, and have professors create the cartridge input data file with information pertinent to the particular activity. There will be one cartridge for every equation that professors wish to take students through, all of which will use the same console.

Third, the equations should be rendered for a more normal appearance. Using JavaScript, we had been forced to use textual approximations of terms such as “pi*r^2” rather than the πr^2 that is preferable. In the final implementation of this activity, we have all equations rendered correctly.

Fourth, we designed a consistent graphical-user-interface. The goal in doing this was to allow students, once accustomed to using any one of the derivation exercises to immediately be able to navigate any of the other derivation exercises.

Java was chosen as the development medium because of the requirements listed above and because Project Links materials are delivered via the Internet. Because of the complexity of graphically rendering arbitrary equations, the Java API WebEQ[5] was incorporated to render the equations graphically.

The system interface was designed with the following components: requirement window, build window, build console, and completed equations (refer again to Figure 1). The requirement window is constructed to house text and equations as specified by the professor, to guide the student to construct the current equation. The build window is constructed much like a calculator screen. Its purpose is to display the equation (the student's answer) as she builds it.

The build console contains the buttons the student uses to construct the equation described in the requirement window. It contains both static and dynamic elements. The dynamic section of the console consists of a 248×121 pixel area allocated for professor-specified buttons. Professors specify the color, size, location, and equation for each button. Professors may have buttons appear and/or disappear between any steps in the equation derivation. For example, if one step in the equation derivation requires $\ddot{x}$, that button can be removed from the console if it is no longer needed for subsequent steps.

The static section of the console consists of three separate blocks. These blocks exist in all activities, and are not modifiable by the professor. The first block contains the arithmetic operators for addition, subtraction, multiplication and division, as well as the left and right parentheses, the square root sign and the ± operator. The second block contains a numeric keypad with numbers zero through nine as well as a decimal point and equals sign. The third block contains control buttons for the equation. **Delete** deletes the last term appended to the equation in the build window. **Clear** clears the build window. **Test** compares the student's answer to the key for that equation and, if the two equations are equivalent, moves the student onto the next step.

Provisions are also included for supplemental help. Dialogue boxes appear when the student presses the **Test** button. For each step in the derivation (each intermediate equation) the professor may specify two messages for the student: one for if she is correct (drawing parallels, making observations), and one for if she is incorrect (a hint on how to go about solving the particular intermediate equation).

Development of the cartridge files has been made as simple as possible. A six-step equation derivation activity required only a 5.5KB text file cartridge. If a professor is familiar with the construction of these activity files (for which there is full documentation) and has an activity storyboarded, it will take him about an hour to code the cartridge file.

Algorithm

To make this system viable, it was necessary to have an extremely robust comparison engine. Previous equation comparison mechanisms were literal, meaning that the student's equation would have to look exactly like the key for it to be marked as correct.

In order to make implementation of the Equation Activity simple for the professors it is desirable for the Activity to need only one equation key in arbitrary form for each step in the derivation process. Furthermore, the system can not issue either false positives (indicating the student's answer is correct when it is not) nor false negatives (indicating the student's answer is incorrect when it is in fact correct).

After analysis of the problem we decided that an organizational comparison engine would be used rather than a numerical comparison engine. Unlike a mathematical tool such as *Maple* or *Matlab*, this engine does not evaluate the expression, it standardizes the order of terms in an equation based on certain properties (listed in the following paragraph). The comparison process consists of passing both the key and the student's equation through this standardization process and then performing a literal comparison on the standardized versions of both equations. If that comparison shows both equations to be literally equal (equivalent to a string comparison), then the student's equation matches the key, and the student is correct and may move on to the next step in the derivation.

Equation Breakdown

Step 1: Check to make sure there are as many opening parentheses as there are closing parentheses. If not the equation is invalid.

Step 2: Insert a multiplication sign between each set of adjacent terms if this module specifies those signs to be assumed. The developer controls this flag, which is specified in the cartridge file. It is generally desirable to assume adjacent multiplication unless working with terms for which this does not make sense, such as terms with summation symbols.

Step 3: Recursively break up the equation into a tree structure. We break up the tree with operators as parent nodes, and the terms on which they operate as their children. For example, the equation "4 + 5" would be encoded as parent node "+" with children "4" and "5". Encoding is done in the reverse of the standard order of operations. In the case of the Project Links Equation Activity, we consider four strata of operators (listed in order of precedence): [=], [+, -, ±], [×, ÷], and [√].

The equation string output by step 2 is taken and broken at the equal sign (if one exists, otherwise it iterates to the next stratum of operators to check for), making its children the terms on either side of the equals sign. The stratum of operators to check for is incremented by one (and rolled over if necessary) and the process is executed on the children of the "=" node. This process is repeated until the entire equation is broken up into its hierarchical organization. Parentheses are taken into account, and dictate the ordering of operations, but are not themselves stored, as their purpose is to dictate order of operations, which is now done by the data structure in which the equation is stored.

Standardizing the Equation

Step 4: Distribute out all terms. For example, transform $4 \times (5 + 6)$ into $(4 \times 5) + (4 \times 6)$ (although this transformation is done to the tree structure). This process is executed until no more terms can be distributed.

Step 5: Distribution of negative signs. Any "-" node with a "+" child will be switched to a "+" node with a "-" child, and signs reversed as appropriate. Any "-" node with a "-"

child will be converted to a "+" node, having its children negated, and its grandchildren upgraded to children. This process is recursive.

Step 6: Percolate out negative signs. For a "×" node, count how many of its children are negative, make all children positive, and if the count (of children which are negative) is an odd number negate the "×" node, otherwise leave it alone. (Changes (-3 × -4 × -5) into -(3 × 4 × 5): again, in tree structure.)

Step 7: Coalesce branches in the tree. For "+" with a "+" child, upgrade all grandchildren of the top "+" node to children, and remove its child "+" node. Execute the exact same process for "×" nodes as well. This transforms structures such as (1 + (2 + 3)) into (1 + 2 + 3).

Step 8: Zero the equation. Add the negative of the right-hand-side of the equation to the left-hand-side, and set the right-hand-side of the equation to zero. This ensures we catch different encodings such as (a + b = c) being the same as (a – c = -b) by moving all terms to one side of the equation.

Step 9: Coalesce branches in the tree. Same as step 7. This must be done again because of new terms introduced into the structure on the left-hand-side of the equation.

Step 10: Recursively sort all nodes. Since the equations will be compared literally, it must be ensured that when we compare "2 + 7 + 3" to "3 + 2 + 7" that they compare exactly. Accordingly, the nodes are sorted (in ascending alphanumeric order.) The way in which they are sorted is irrelevant as long as they are sorted consistently in the same manner.

Step 11: Make negative sign uniform. If the first term on the left-hand-side is negative, all terms on the left-hand-side are negated. If not, then the left-hand-side is left alone. This is done to ensure the first term (after having been sorted) is always positive. For example, equations (4 + x = 0) and (-4 - x = 0) are both correct, but encoded differently. This step forces encoding in the manner of the first (4 + x = 0).

This encoding was incorporated into the (custom-built) *Equation* object in Java. The encoding of the equation takes place automatically in the object's constructor. When it comes time to compare the student's equation against the key, the student's equation is translated (by creating a new *Equation* object with the user's equation as the input argument), and then compared using the .equals(*Equation*) method (of object *Equation*) which compares one equation against another (in our case comparing the student's equation against the previously-stored equation keys) and returns true or false, indicating a successful comparison. Both equation trees are read through, and must now (being in the standardized form) be of exactly equal size and content in exactly the same order. If this is true then the equations are equivalent and the student is correct.

Summary

It can prove helpful in teaching mathematical concepts to make students aware of the derivation or solution of an equation, and of the sequence of steps taken to attain that final equation. This equation activity (functionally a Java applet) provides a method to implement that teaching strategy. It is generalizable such that it may be used for any mathematical concept, limited only by the fixed size real-estate of the equation containers.

Acknowledgements

Project Links gratefully acknowledges the support of Rensselaer and of the National Science foundation under grant #DUE-9552465.

The Equation Activity was first conceived by Billie J. Rinaldi, a graduate student in Mathematics at Rensselaer, and a key developer of the Mechanical Oscillations series of modules.

The talents of Daniel M. Manthey, a Rensselaer graduate student in Computer Science, were instrumental in the successful coding of the Equation Activity.

KENNETH S. MANNING, PhD
Ken Manning is the Technical Manager for Project Links, and an Adjunct Associate Professor for the Core Engineering Program at Rensselaer Polytechnic Institute in Troy, New York. He has also worked as a thermal-hydraulic design engineer for General Electric, first at the Knolls Atomic Power Laboratory, and then at the Corporate Research & Development Center. His B.S. is in Physics from the University of Oregon, received in 1976, his M.S. is in Mechanical Engineering from the University of Illinois at Chicago in 1984, and his Ph.D., also in Mechanical Engineering, is from Rensselaer in 1992.

LUKE B. BELLANDI
Luke Bellandi is a recent graduate from Rensselaer majoring in Electrical & Systems Engineering. He has been a programmer with Project Links for three years, and has also contributed to the work of the Academy of Electronic Media at Rensselaer. He has self-published a CD of original music. He has accepted a position with Apple in Cupertino, California.

Bibliography

1 The Project Links Web site, http://links.math.rpi.edu/

2 Manning, K., “Project Links: Interactive Web-Based Modules for Teaching Engineering”, ASEE Annual Conference and Exposition, St. Louis, Missouri, Session 2620, 2000.

3 NEEDS: The National Engineering Education Delivery System, http://www.needs.org/

4 Fisher, T., Orr, J., & Scott S., “Randomized Interval Analysis Checks for the Equivalence of Mathematical Expressions”, preprint received directly from J. Orr, 2000.

5 WebEQ by Design Science, Inc. is available at http://www.mathtype.com/webmath/.