THE UNIVERSITY OF AKRON
Theoretical and Applied Mathematics

Extending the Exerquiz Package Special Processing

D. P. Story

Needs revision

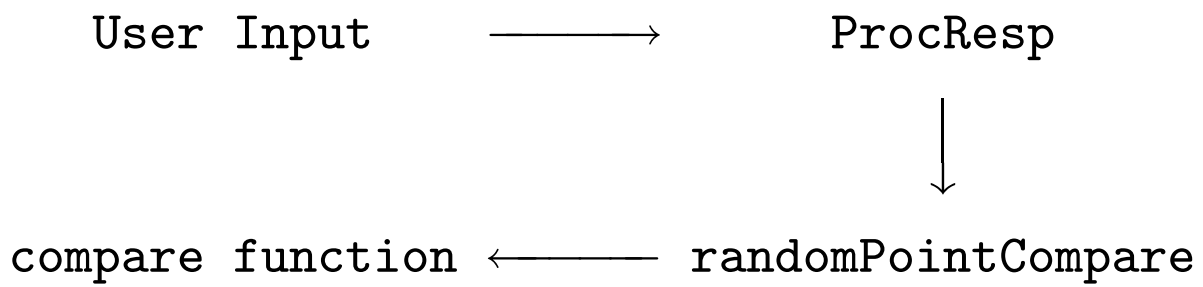
Last Revision Date: July 24, 2009
dpstory@uakron.edu
Version 3.0

Table of Contents

1. Introduction

The command `\RespBoxMath` is used to define a math fill-in question. Normally, such a question takes a numerical answer, or takes a function of a single variable, x, that reduces to a number when x is given a value.

When the user enters an answer into the response box, a document-level JavaScript function, `ProcResp` is called to process the answer. This function then calls another JavaScript function called `randomPointCompare`, which randomly chooses points from a specified interval. For each of the points chosen, `randomPointCompare` then calls the (default) compare function to compare the author's answer with the user's answer. The diagram below illustrates the normal program flow for processing math fill-in questions.



Recently, `\RespBoxMath` and `ProcResp` have been rewritten to allow

the author to modify this normal program flow. By specifying some optional arguments in the `\RespBoxMath` command, the author can call a custom response function and use a custom compare function as well. The sections that follow discuss the methods of redirecting program flow and of writing new compare and response functions. An extensive example is developed.

2. \RespBoxMath

The `\RespBoxMath` now has a ten parameters that can be used to modify the default behavior of processing the user's input. Here is the syntax:

```
\RespBoxMath[#1]#2(#3)[#4]#5#6#7#8[#9]*#10
```

Parameters:

`#1`: Optional parameter used to modify the appearance of the text field.

`#2`: The correct answer to the question. This must be a numerical value, or a function of one variable. JavaScript Note: In

JavaScript, functions such as `sin(x)` and `cos(x)` are methods of the `Math` object. It is not necessary, however, to type `Math.sin(x)` or `Math.cos(x)`; this is done by inserting the expression into a `with(Math)` group.

`#3:` An optional parameter, *delimited by parentheses*, that defines the independent variable; `x`, is the default value. Note that this parameter is set off by parentheses. For a multivariate question, just list the variables in juxtaposition, `(xyz)`.

`#4:` Optional, a named destination to the solution to the question. If this parameter appears, then a solution must follow the question, enclosed in a `solution` environment. There is an alternate forth parameter, a '`*`'. In this case, an automatic naming scheme is used.

`#5:` The number of samples points to be used, usually 3 or 4 is sufficient.

`#6:` Precision required, the ϵ value, if you will.

`#7:` Parameters `#7` and `#8` are used to define the interval from which to draw the sample points. There are two forms: (1) `#7` is the

left-hand endpoint of the interval and #8 is the right-hand endpoint (the use of #7 and #8 in this form is deprecated); (2) the interval is defined by standard interval notation, [a,b]. For a multivariate question—one where parameter #2 lists more than one variable, separate the intervals for each variable by a 'x', [0,2]x[1,2]x[3,4]. Here, 'x' stands for Cartesian Product.

#8: (1) Parameter #8 is the right-hand endpoint of the interval (the use of this parameter is deprecated); (2) in the second case, #8 is not used.

#9: This optional parameter is the name of a customized comparison function.

#10: (Only detected if following an asterisk, '*') The name of a JavaScript function that is to be used to process the user input.

Parameters #9 and #10 can be used to specify custom *compare* and *response* functions, respectively. The compare function is discussed in Section 3 followed by a discussion of the response function in Section 5.

You can write your own compare and response functions. An example of a custom *compare* function is given in the file `jquiztst.tex` as well as in the preamble of this document; this file also demonstrates a custom *response* function. These JavaScript functions are defined using the insdljs package.

3. The Compare Function

The way exerquiz determines whether a user's response to a math fill-in question is by randomly choosing a number of points (parameter #5) from an interval of numbers (parameters #7 and #8) and comparing the user's answer with the author's provided answer (parameter #2). If at any of the comparisons, the two answers differ by more than some acceptable value (parameter #6), the user's answer is judged *incorrect*.

The listing of the default compare functions follows:

```
function diffCompare(_a,_c,_v,_F,_G) {
    var aXY = _c.split(",");
    var n = aXY.length
    with(Math) {
```

```
    for (var i=0; i< n; i++)
            eval ( "var "+_v[i] + " = " + aXY[i] + ";");
    _F = eval(_F);
    if ( app.viewerVersion >= 5)
    {
        var rtnCode = 0;
        eval("try { if(isNaN(_G = eval(_G))) rtnCode=-1; }"
            + " catch (e) { rtnCode=1; }");
        switch(rtnCode)
        {
            case  0: break;
            case  1: return null;
            case -1: return -1;
        }
    }
    else
        if(isNaN(_G = eval(_G))) return -1;
    return abs ( _F - _G );
    }
}
```

Where '_a' is an string of comma delimited numbers representing the endpoints of the intervals from which to sample points; the param-

eter '_c' is a comma delimited string of randomly chosen point(s); the parameter '_v' is a string listing the independent variables of the questions; '_F' is the author's answer, and '_G' is the user's answer. (Note: We give these parameters special names, beginning with an underscore, to protect the function from the user. Without these underscores, it is possible for the user to enter an answer such as 'cos(x) + c the value of 'c' has been passed to function from the calling function. When we perform the eval(G), the value for 'c' will be substituted in; this might lead to unexpected results. It is unlikely, however, that the user will enter an expression like 'cos(x) + _c. With the underscores, 'c' is undefined, and an exception will be thrown.)

JavaScript 1.5, the core JavaScript used by Acrobat 5.0, now throws an exception if an error occurs. (Prior versions of JavaScript do not.) Typically, you can try to catch any critical error, for better error handling. Notice the line

```
eval("try { if(isNaN(_G = eval(_G))) rtnCode=-1; }"
    + " catch (e) { rtnCode=1; }");
```

We make the evaluation of _G within a try/catch Recall that expres-

sion the user enters is passed to this function as the parameter `_G`. If the user has entered any grossly incorrect expression, an exception will be thrown when we try to evaluate it at `_c`.

In Acrobat 4.0–4.05, `try/catch` are reserved words and cannot be used. Therefore, I have made the relevant code into a string which gets evaluated, in the case the user is viewing the document in Acrobat Reader 5.0 or above. If a Reader prior to 5.0 is being used, the reserved words are in a string, and are not parsed by the JavaScript interpreter.

To understand the code, consider a simplified form of the `try/-catch`:

```
try { if(isNaN(_G = eval(_G))) return -1; }
catch (e) { return null; }
```

consider the following three examples.

▶ Example 1: User enter a good expression:

```
var x = 4;
var _G = "Math.sqrt( x )";
try {
    if(isNaN(_G = eval(_G))) app.alert("-1");
```

```
        else app.alert("_G = " + _G);
} catch (e) { app.alert("null"); }
```

If this script is executed, `_G = 2` appears in the alert box.

▶ Example 2: User enters a functions whose domain falls outside the domain of the correct answer. Suppose the user enters the expression, `sqrt( -x )`.

```
var x = 4;
var _G = "Math.sqrt( -x )";
try {
    if(isNaN(_G = eval(_G))) app.alert("-1");
        else app.alert("_G = " + _G);
} catch (e) { app.alert("null"); }
```

When this script is executed, -1 is appears in the alert box; i.e., `iNaN` returned a `true`. No exception was thrown here. This is considered an error. The user entered a function the domain of which did not contain the range of numbers that are to be tested.

▶ Example 3: User enters an express that throws an exception:

```
var x = 4;
var _G = "Math.sqrt( t )";
```

```
try {
    if(isNaN(_G = eval(_G))) app.alert("-1");
        else app.alert("_G = " + _G);
} catch (e) { app.alert("null"); }
```

The declaration of `_G` has been changed to `"Math.sqrt( t )"`. Now, when we try to evaluate `G`, an exception is thrown, and `null` appears in the alert dialog. The error message generated by this exception is `ReferenceError: t is not defined`. This kind of error is considered a typo, and the user is not penalized.

3.1. The `randomPointCompare` Function

The compare function is called by `randomPointCompare`. It is this function that randomly chooses a number of points from the specified interval, then calls the compare function to evaluate and compare the correct answer with the user's answer. It is `randomPointCompare` that processes the return value of the compare function. Below is a snippet of `randomPointCompare` so you can see how it processes the return value of the compare function.

```
error = comp(a,cXY,indepVar,CorrAns,userAns);
```

```
if (error == null)
    return null;                              // this is considered a typo
if ( (error == -1) || (error > epsilon) )
{
    j=-1;                                     // j=-1 signals an error
    break;
}
```

4. A Custom Compare Function

The custom compare function in the preamble of this document does a compare appropriate to comparing two indefinite integrals. We list the version of this compare function that uses `try/catch`, Acrobat Reader 5.0 or later is needed.

```
function indefCompare(_a,_c,_v,_F,_G) {
    var eqC;
    var aAB = _a.split(",");
    var aXY = _c.split(",");
    var n = aXY.length
    with(Math) {
        for (var i=0; i< n; i++)
            eval ( "var "+_v[i] + " = " + aAB[2*i] + ";" );
```

```
% var C = 0 is used to designate an arbitrary constant
        var C = 0;
        if ( app.viewerVersion >= 5)
        {
            var rtnCode = 0;
            eval("try {"
                + " if(isNaN(eqC = eval(_F)-eval(_G))) rtnCode=-1;}"
                + " catch (e) { rtnCode=1; }");
            switch(rtnCode)
            {
                case  0: break;
                case  1: return null;
                case -1: return -1;
            }
        }
        else
            if (isNaN(eqC = eval(_F)-eval(_G))) return -1;
        for (var i=0; i< n; i++)
            eval ( "var "+_v[i] + " = " + aXY[i] + ";" );
        _F = eval(_F);
        if ( app.viewerVersion >= 5)
        {
            var rtnCode = 0;
```

```
            eval("try { if(isNaN(_G = eval(_G))) rtnCode=-1; }"
                + " catch (e) { rtnCode=1; }");
            switch(rtnCode)
            {
                case  0: break;
                case  1: return null;
                case -1: return -1;
            }
        }
        else
            if(isNaN(_G = eval(_G))) return -1;
        return abs( _F - _G - eqC );
    }
}
```

First, we evaluate each at the left-hand endpoint (`_a`) and store this value as `eqC`. Then we compare the two function `_F` and `_G` and the randomly supplied point `_c`. We then return the `abs(_F - _G - eqC)`. If the two supplied are both correct, they would differ by a constant.

Here is an example of usage of this custom compare function:

▶ $\int \sin(x)\,dx =$ [ ] Ans [ ] Clear

5. The Response Function

The *response* function is a JS function that is called when the math fill-in data is committed by the user. Basically, it strips out all white space; calls `ParseInput()` (which scans the user input for syntax errors, converts exponents to the `pow()` functions, etc.); calls `randomPointCompare()` which returns `true` or `false` depending on whether the user's answer was close enough to the author's answer; and finally, notifies the field of the result (calls `notifyField()`).

The default response function is the JavaScript `ProcResp()`, but this function can be replaced by a "custom response function", such as the one in the preamble of this document. The listing of `ProcResp()` follows:

```
function ProcResp(flag,CorrAns,n,epsilon,a,indepVar,comp)
{
    if (!ProcessIt) return null;
    ok2Continue = true;
    var success;
    var fieldname = event.target.name;
    var UserAns = event.value;
    CorrAns = ParseInput(CorrAns);
```

```
    if (!ok2Continue) {
        app.alert("Syntax error in author's answer!"
            + " Check console.", 3);
        console.println("Syntax Error: " + CorrAns);
        return null;
    }
    UserAns = ParseInput(UserAns);
    if (!ok2Continue) return null;
    success=randomPointCompare
        (n,a,indepVar,epsilon,CorrAns,UserAns,comp)
    if ( success == null )
        { app.alert(\eqSyntaxErrorUndefVar,3); return null; }
    return notifyField(success, flag, fieldname);
}
```

`ProcResp` takes eight arguments, `flag, CorrAns, n, epsilon, a, b, indepVar, comp`. The `flag` indicates whether we are in 'silent mode' (no immediate reporting back of results); obviously, `CorrAns` is the author's correct answer; `n` is the number of points to sample; `epsilon` is the acceptance level; arguments `a` and `b` are the left and right hand endpoints, respectively; `indepVar` is a string indicating the independent variable, the default is `"x"`; and finally, `comp` is the name

of the compare function that is to be called.

The JS function `changeToX` uses regular expressions to replace the declared `indepVar` with an internal variable. After changing variables, we call `randomPointCompare`, then `notifyField` of the results.

▶ Important: Any custom response function must use the above eight arguments, and finish off by returning a `true` or `false` value. In the above example, the return value of `notifyField` is used.

6. Demo: Quizzes

We illustrate the `ProcVec` and `indefCompare` function through a `shortquiz` and `quiz` by extending `exerquiz` to be able to process vector answers. Other extensions are possible, such as processing functions of several variables, for example.

▶ Click on the "Ans" button to see the answer. If the "Ans" button has a green boundary, that problem has a solution. Shift-click on "Ans" to jump to the solution.

6.1. A shortquiz

Instructions: Enter vectors with angle brackets, e.g., `<1,2,3>`. You can also enter scalar multiples of vectors, e.g., `4*<1,2,3>`.

Quiz Let $\vec{a} = \langle 1, 2, 3\rangle$, $\vec{b} = \langle 3, 2, 1\rangle$ and $\vec{f}(t) = \langle e^t, t^2, \sin(t)\rangle$. Calculate each of the following.

1. $\vec{a} + \vec{b} =$ [ ] Ans

2. $\vec{a} \cdot \vec{b} =$ [ ] Ans

3. $\vec{a} \times \vec{b} =$ [ ] Ans

4. $\vec{f}'(t) =$ [ ] Ans

5. $\displaystyle\int \vec{f}(t)\, dt =$ [ ] Ans

Clear

6.2. A quiz

Instructions: Same instructions as in the `shortquiz`.

Begin Quiz Let $\vec{a} = \langle 1, 2, 3 \rangle$, $\vec{b} = \langle 3, 2, 1 \rangle$ and $\vec{f}(t) = \langle e^t, t^2, \sin(t) \rangle$. Calculate each of the following.

1. $\vec{a} + \vec{b} =$
2. $\vec{a} \cdot \vec{b} =$
3. $\vec{a} \times \vec{b} =$
4. $\vec{f}'(t) =$
5. $\int \vec{f}(t)\, dt =$

End Quiz Score: Correct

Answers:

Solutions to Quizzes

Solution to Quiz:

$$\int \sin(x)\,dx = -\cos(x) + C$$

Solution to Quiz: We load the vectors

$$\vec{a} =< 1, 2, 3 > \text{ and } \vec{b} =< 3, 2, 1 >$$

into a 3×3 determinant, like so:

$$\begin{aligned}
\vec{a} \times \vec{b} &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 2 & 3 \\ 3 & 2 & 1 \end{vmatrix} \\
&= \begin{vmatrix} 2 & 3 \\ 2 & 1 \end{vmatrix} \vec{i} - \begin{vmatrix} 1 & 3 \\ 3 & 1 \end{vmatrix} \vec{j} + \begin{vmatrix} 1 & 2 \\ 3 & 2 \end{vmatrix} \vec{k} \\
&= (2-6)\vec{i} - (1-9)\vec{j} + (2-6)\vec{k} \\
&= -4\vec{i} + 8\vec{j} - 4\vec{k} \\
&=< -4, 8, -4 > \\
&= 4 < -1, 2, -1 >
\end{aligned}$$

Solution to Quiz: We simply integrate componentwise:

$$\begin{aligned}\int \vec{f}(t)\,dt &= \int \langle\, e^t, t^2, \sin(t)\,\rangle\,dt \\ &= \langle \int e^t\,dt, \int t^2\,dt, \int \sin(t)\,dt\rangle \\ &= \langle e^t, \tfrac{1}{3}t^3, -\cos(t)\rangle + \vec{C}\end{aligned}$$

In the syntax of this document, the answer is `<e^t,t^3/3,-cos(t)>`.

Solution to Quiz: We load the vectors

$$\vec{a} =< 1, 2, 3 > \text{ and } \vec{b} =< 3, 2, 1 >$$

into a 3×3 determinant, like so:

$$\begin{aligned}
\vec{a} \times \vec{b} &= \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & 2 & 3 \\ 3 & 2 & 1 \end{vmatrix} \\
&= \begin{vmatrix} 2 & 3 \\ 2 & 1 \end{vmatrix} \vec{i} - \begin{vmatrix} 1 & 3 \\ 3 & 1 \end{vmatrix} \vec{j} + \begin{vmatrix} 1 & 2 \\ 3 & 2 \end{vmatrix} \vec{k} \\
&= (2-6)\vec{i} - (1-9)\vec{j} + (2-6)\vec{k} \\
&= -4\vec{i} + 8\vec{j} - 4\vec{k} \\
&=< -4, 8, -4 > \\
&= 4 < -1, 2, -1 >
\end{aligned}$$

Solution to Quiz: We simply integrate componentwise:

$$\begin{aligned}\int \vec{f}(t)\,dt &= \int \langle\, e^t, t^2, \sin(t)\,\rangle\,dt \\ &= \langle \int e^t\,dt, \int t^2\,dt, \int \sin(t)\,dt\rangle \\ &= \langle e^t, \tfrac{1}{3}t^3, -\cos(t)\rangle + \vec{C}\end{aligned}$$

In the syntax of this document, the answer is `<e^t,t^3/3,-cos(t)>`.