Maple T.A. implementation tips

# Introduction

Maple T.A. is an online assessment system that was designed especially for courses that involve mathematics. It uses the Maple engine to grade the students answers and has a complete grading report system.

On a basic level, it is simple to implement stationary questions (i.e. questions that look the same every time they are opened). On a higher level, algorithms can be programmed to randomize questions and open more sophisticated possibilities in grading. Also on a higher level, Math Apps can be used to generate sketching questions with very flexible grading possibilities.

My goal in writing this document is to help in understanding the questions and algorithms I used and developed in the last years for the Coastal Dynamics I course. I will focus on the randomization of questions and on the use of Math Apps.

# Choosing the type of question

Different types of questions are available to use in Maple T.A.

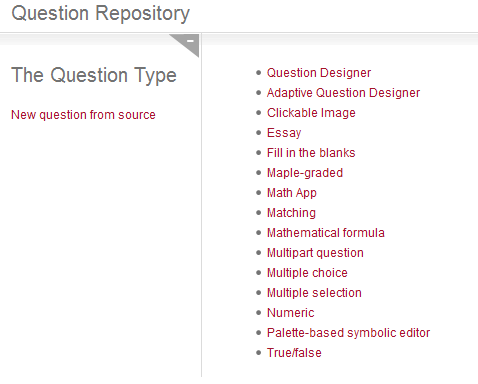


Figure 1 – Options available in the “New Question” page.

With the exception of “Clickable Image” all the options in the list shown in the “New Question” page (Figure 1) can be also implemented from the “Question Designer”. This type of question is more flexible because the type of answer area can be changed easily. In the “Question Designer” type of question it is also possible to have sub-questions. In other question types the answer area is practically locked and one must go into the Source code of the question to change it (and lose some data in the process). In most cases my preference is to use the “Question designer”.

Up to now we didn’t use the “Adaptive Question Designer”. This question type in which the student is presented with one subquestion, or section, at a time, and the question behavior depends on whether the student answers the subquestion correctly.

# Randomization

In Maple T.A. questions can be randomized, i.e., variables can assume different values, multiple choice questions can present different options, figures can change, etc.

For simple randomizations, there is an “Algorithm Designer” that can be of use to create variables. However, for more complex algorithms other functions are necessary. Maple T.A. has a number functions directly available and more complex functions are also available using Maple commands (commands from the Maple language). In my experience, I needed only 2 or 3 basic functions and a Maple command to randomized most of the questions. I will introduce them as needed when I explain the algorithms.

## Simple randomization example

In this first example we build a question where a word changes in the question and the answer changes accordingly. The question text is:

Is 1 $word 0?

In Maple T.A. all variables must start with the prefix $. In this question we want the variable *$word* to be one of the following options: “bigger than”, “smaller than” and “equal to”. The algorithm for this is:

$choice=rint(3);

$word=switch($choice,”bigger than”, “smaller than”, “equal to”);

In this code, two of the most used functions appear:

1. rint(n), with *n* being an integer, returns a random integer smaller than *n*, i.e., 0,1 or 2.
2. switch(n, alternative1, alternative2, alternative3), with *n* being an integer and the alternatives being numbers or strings. It returns the alternative with the index *n*. In this case if *n*=1, alternative1 is returned, if *n*=2, alternative2 is returned and so on. switch can have any number of alternatives.

To create the response area click the “Insert/Edit Response Area” button (Figure 2). In the “Edit Response Area” dialog (Figure 3) the answer options are defined, as well as other preferences. In this case the answer options are only “Yes” and “No”. Clicking the OK button takes us to the dialog where the correct value(s) is/are set. In this case an algorithmic value (a variable) must be provided. In this case we will call it $answer. In this case, this variable can take the values 1, if the answer is “Yes” and 2, if the answer is “No” and will be defined in the algorithm as:

$answer=switch($choice,1,2,2);

This means that if $choice is 0, $word is “bigger than” and $answer is 1 (“Yes”). If $choice is 1 or 2, $word is “smaller than” or “equal to” and $answer is 2 (“No”).

There are always different ways to implement a question. This example, and most questions that use algorithms can be implemented in different ways.

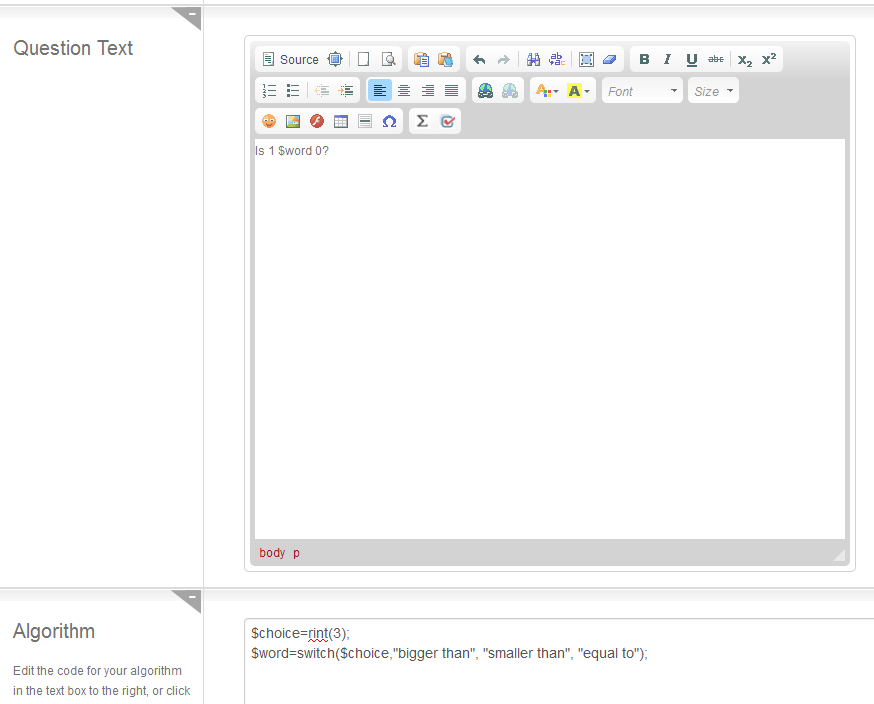


Figure 2 – Insert/Edit response area button.

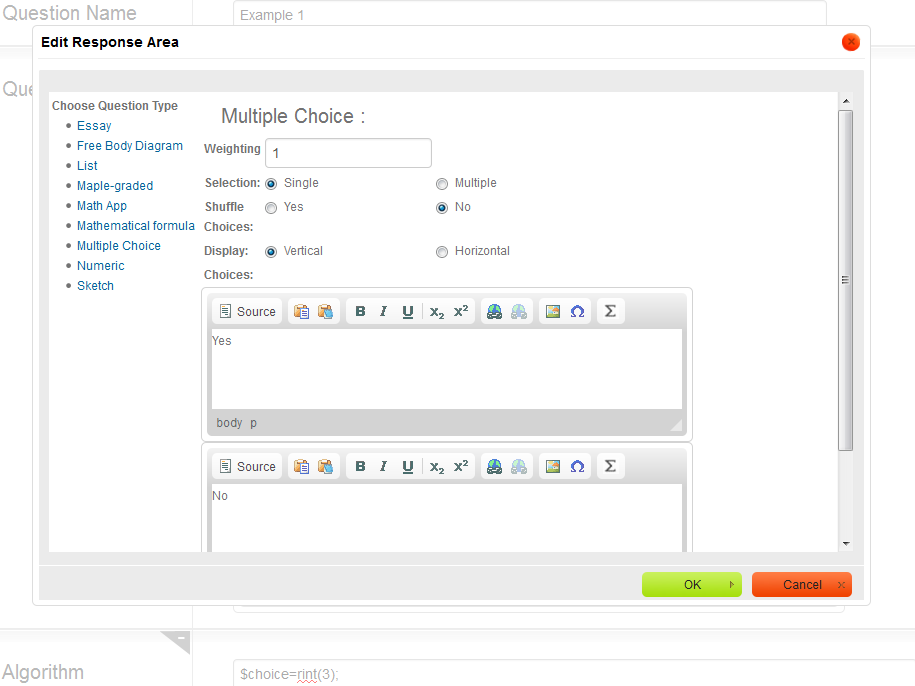


Figure 3 – Response Area dialog.

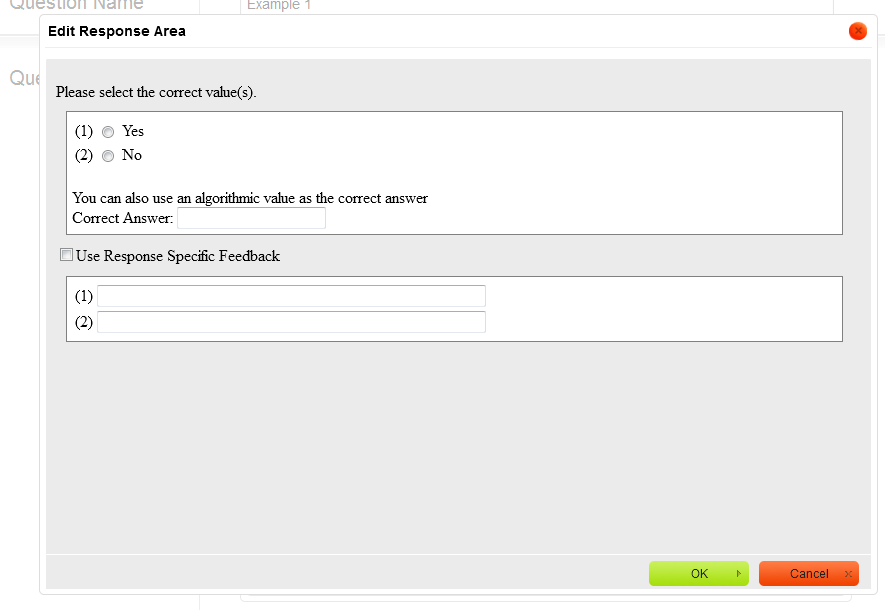


Figure 4 – “Select correct values” dialog.

## Multiple selection with a varying list of options

The randomization can be also used to present different alternatives in each realization of the question. One of the most useful randomization technique is based on shuffling a list. For example, in question ”Measures against episodic structural erosion 2 - Test B” three options are chosen randomly for the three sub-questions.

The algorithm shuffles the list: (0,1,2,3,4,5,6) with the command.

$A=maple("randomize():StringTools:-Randomize():combinat[randperm]([0,1,2,3,4,5,6])");

Then the first three indexes are retained (in this case the indexes could be,e.g, 4,0 and 6):

$index1 =maple("$A[1]");

$index2 =maple("$A[2]");

$index3 =maple("$A[3]");

Each index corresponds to an option:

$str1 = "Groynes";

$str2 = "Emerged detached breakwater";

$str3 = "Nourishment";

$str4 = "By-pass system";

$str5 = "Jetties";

$str6 = "Sea dike";

$str7 = "Revetment";

For each sub-question the selected option is assigned to the corresponding variable (if index1=4 then $term1 would be equal to “jetties”[[1]](#footnote-1)):

$term1=switch($index1,"$str1","$str2","$str3","$str4","$str5","$str6","$str7");

And the corresponding answers (here the answers are given as the number of the correct option). If $index=4 the correct answer is 1, which corresponds to “both structural and episodic erosion”

$ans1 =switch($index1,1,3,3,1,1,2,2);

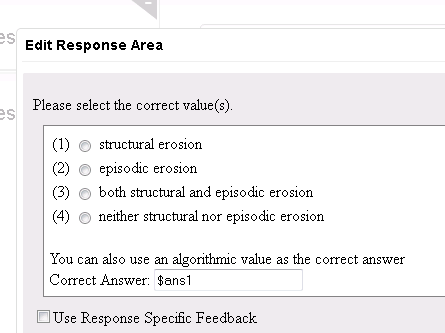


Figure 5 - $ans1 variable defining the correct answer.

The same procedure is done for the other 2 options.

## Varying list of options with variable question

It is possible to combine the previous two types for a deeper randomization. An example is question “Beach states - general - Test B”. Also in this question another solution for the definition of the answers is used. Instead of indicating which option is correct with a number as in Figure 5 the options are variables themselves (Figure 6).

In this question the student is asked about the typical characteristics of a reflective or dissipative beach. After shuffling the index list, as in the previous example, a list of antagonistic features is defined: $a1, $a2, etc are features of reflective beaches and $b1, $b2, etc, features of dissipative beaches.

The correct answers (e.g., $right1) are defined with a switch statement similar to the one used in the previous question inside another switch statement that chooses between “dissipative” and “reflective”:

$right1=switch($choice,switch($indexg1,"$a1","$a2","$a3","$a4","$a5"),switch($indexg1,"$b1","$b2","$b3","$b4","$b5"));

The wrong answers (e.g., $wrong1) are defined in the same way, but using the logical not() operator in the first switch in order to choose the wrong answer.

$wrong1=switch(not($choice),switch($indexf1,"$a1","$a2","$a3","$a4","$a5"),switch($indexf1,"$b1","$b2","$b3","$b4","$b5"));

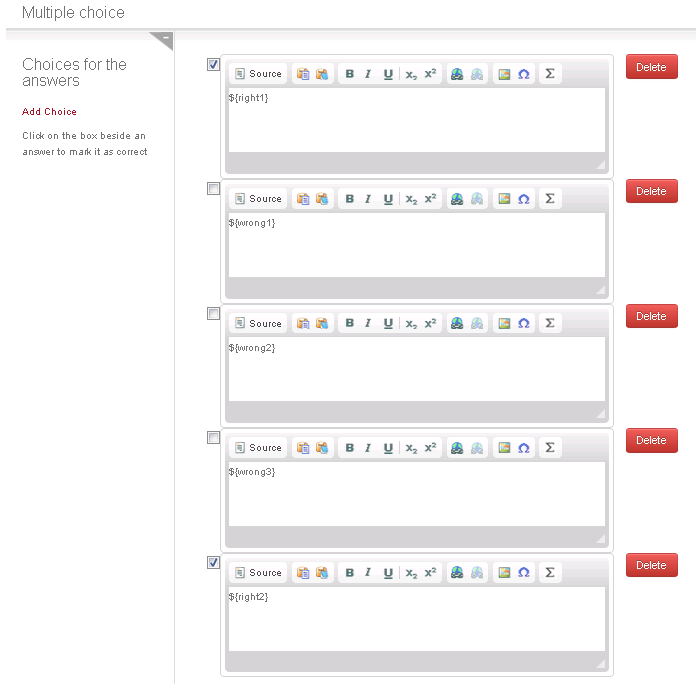


Figure 6 - defining the answers with variables.

## Numeric questions

In a numeric question the students need to fill in a value in the answer field. The value can be a variable defined in the algorithm. There are two types of questions that can be used for this purpose: Numeric and Maple Graded (Figure 7 and Figure 8 ). The former is more simple and straight-forward. The later more flexible but also more complicated.

A good example of the use of these question types is Question “Groyne system 1”. It was initially developed as a Numeric question with one right answer defined in the algorithm as:

$fact1=rint(5);

$fact2=rint(5);

$angle1=20-$fact1;

$angle2=10-$fact2;

$answer1=maple("evalf(($angle1-$angle2)/$angle1)");

A few options for the grading of the question can be tweaked (Figure 7). These options are self-explanatory. One that we use in almost all numeric questions is “Accept arithmetic”. This question allows the student to use a formula as an answer.

The error tolerance can be set in the “Required with” drop-down list. The options are self-explanatory..

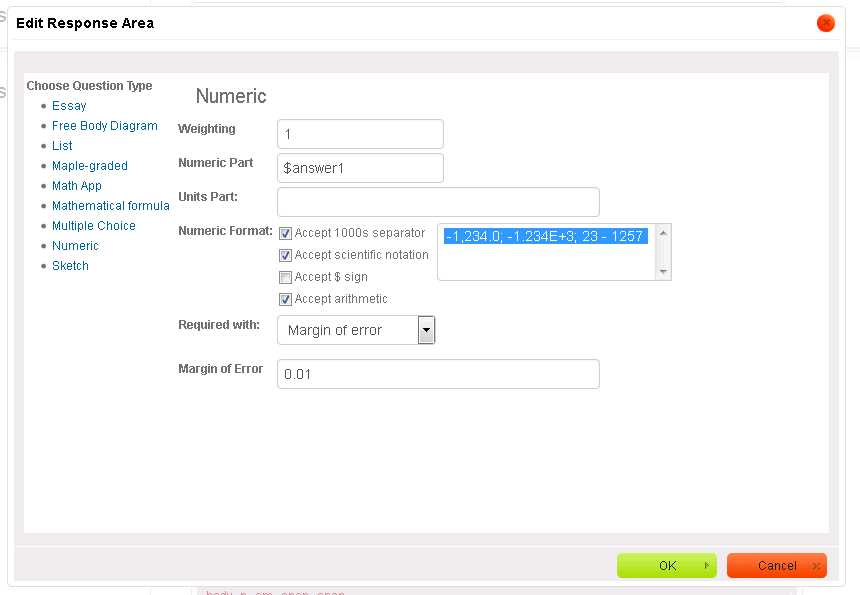


Figure 7 – Numeric question dialog.

After a student pointed out that another answer was also valid we needed to change the question in order to accept two answers. This can be done with a “Maple-graded” type question. In this question a Maple condition is evaluated. If the condition is true the grade is 1 (correct answer) and if it’s false the grade is 0 (incorrect answer). It is not possible to get an intermediate grade (0.5 for example).

In the Groyne system question we added the second accepted answer to the algorithm:

$answer2=maple("evalf(sin(2\*($angle1-$angle2)\*Pi/180)/sin (2\*$angle1\*Pi/180))") ;

In the Grading code we compare the students answer ($RESPONSE) to the two accepted answers and give 1 point if one of the conditions is correct:

is(abs($answer1-evalf($RESPONSE))<= 0.01) or is(abs($answer2-evalf($RESPONSE))<= 0.01);

Note that if only the first condition is used: is(abs($answer1-evalf($RESPONSE))<= 0.01);, the question is equivalent to the numerical version described above.

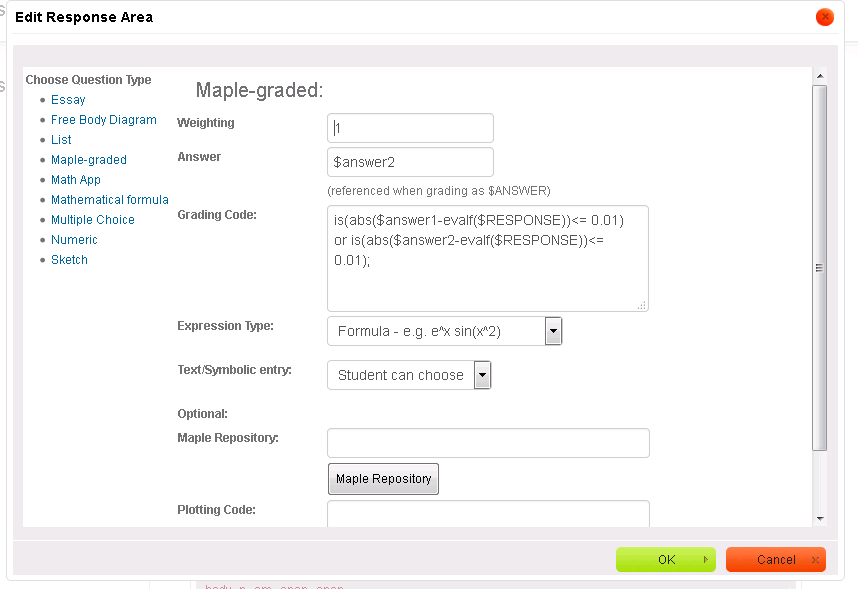


Figure 8 – Maple-graded question dialog.

# True and false questions

There is a specific True and False question type but for the reasons explained above we use a multiple choice for this effect.

For a simple True and False one needs only to include one “Multiple Choice” answer area with choices True and False.

With a little “hack” the partial grading of multiple choice questions can be used with series of true and false questions to penalize students that try to guess an answer. For example if we have Statement 1 and Statement 2 the options presented to the student would be:

* Statement 1: True
* Statement 1: False
* Statement 2: True
* Statement 2: False

The question will be correct only if the student has both conditions right. In this case, the best strategy for a student that only knows one of the answers is to leave the other unanswered, and for that the student gets half of the points.

# Math App questions

Math Apps are questions that take advantage of a Maple application. The input of the students to this application is recorded and graded. The only Math App question now we use is a Sketch question developed by Metha Kamminga.

## Math App – Sketch

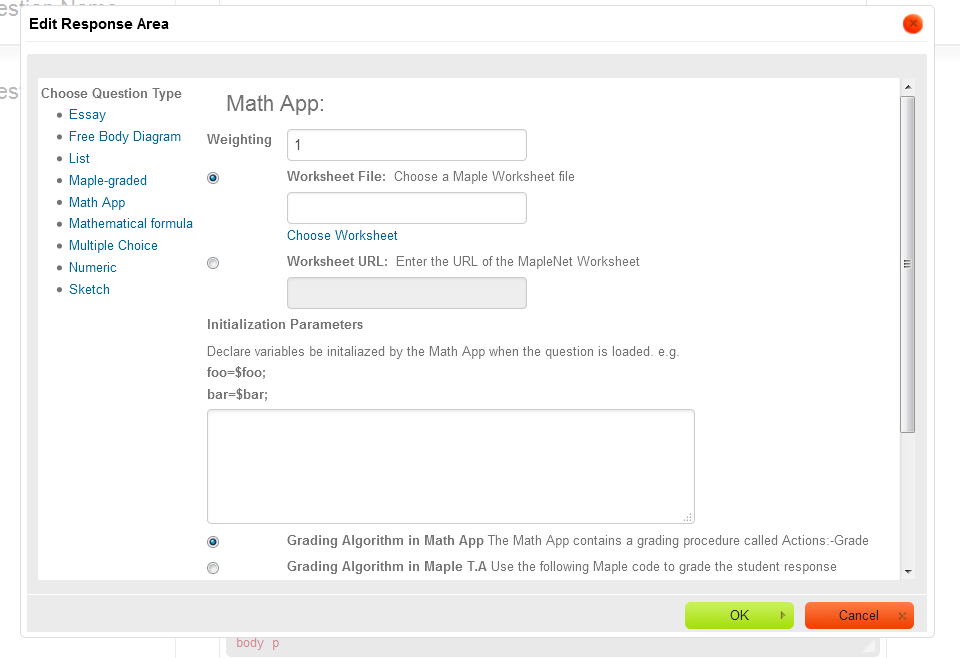
In this type of question, students draw a line by placing points on a Cartesian axis. The Math App’s function is defined on a Maple Worksheet. You can see the code if you open the worksheet file in Maple and you can test the Math app by running the code.

One doesn’t need to change the worksheet in order to use it in Maple TA. It’s parameters (e.g. axes ranges, labels, etc) are accessible from the Math App dialog in the question designer. In this section I present the code for the grading and for the algorithm. I present it first without comments so you can easily copy and paste it from the Word document or pdf, and then I explain what each line does.

### Building a Math App Sketching question

The following steps are necessary to build a Sketching question:

* Create a “Question designer” question
* Insert/Edit response area button.
* In the “Edit response area” dialog choose the Math App question type
* Choose a worksheet file. We currently have two types of worksheet: “SketchApp” and “SketchAppExtraGrid”. The differences will be explained later. The worksheets are in the “MethaMW” directory.



* Initialize parameters. Copy and paste the following code in the text field For the “SketchApp” worksheet:

listranges=$listranges;

horlabel=$horlabel;

vertlabel=$vertlabel;

height=$height;

width=$width;

npt=$npt;

grid=$grid;

Bgcolor=$Bgcolor;

BgPoints=$BgPoints;

Bgrangespline=$Bgrangespline;

Bgf=$Bgf;

Bgrangef=$Bgrangef;

Bgg=$Bgg;

Bgrangeg=$Bgrangeg;

And for the “SketchAppExtraGrid” worksheet:

listranges=$listranges;

horlabel=$horlabel;

vertlabel=$vertlabel;

height=$height;

width=$width;

npt=$npt;

grid=$grid;

Bgcolor=$Bgcolor;

BgPoints=$BgPoints;

Bgrangespline=$Bgrangespline;

Bgf=$Bgf;

Bgrangef=$Bgrangef;

Bgg=$Bgg;

Bgrangeg=$Bgrangeg;

gridx=$gridx;

factorgridx=$factorgridx;

gridy=$gridy;

factorgridy=$factorgridy;

* Select the “Gradin Algorithm in Maple T.A.”option
* Enter the Maple Code to grade the student response. An example is given below I will try to explain step by step what the code does:

pointlist := DocumentTools[GetProperty](MathContainer0, expression):

if nops(pointlist) < 3 then return 0 else

spl:=CurveFitting[Spline](pointlist, x):

erro:=0:

grade:=0:

for pt in $antw1 do

erro:=erro+abs(pt[2]-eval(spl,x=pt[1]));

end do;

if erro<=0.4 then grade:=grade+0.5;

end if;

if (is(eval(diff(spl, x),x=-3), negative)) and (is(eval(diff(spl, x),x=-1), positive)) and (is(eval(diff(spl, x),x=1), positive)) and (is(eval(diff(spl, x),x=3), negative)) then grade:=grade+0.5;

end if;

return grade;

end if;

Explanation of the code:

pointlist := DocumentTools[GetProperty](MathContainer0, expression):

Gets the list of points placed by the student

if nops(pointlist) < 3 then return 0 else

If the student placed less than 3 points the grade is zero (Math App returns a zero value). This condition was included to avoid errors when reloading Math App questions.

spl:=CurveFitting[Spline](pointlist, x):

Builds the spline representation of the list of points placed by the student. This spline is what is seen by the student in the Sketch. The spline will be what is used in the grading process.

erro:=0:

grade:=0:

Initializes variables: “erro” will be accumulating the errors calculated at each solution point (later I will explained how this solution is introduced) and “grade” will be the value that is returned representing the final grade of the question.

for pt in $antw1 do

erro:=erro+abs(pt[2]-eval(spl,x=pt[1]));

end do;

This is a for loop that evaluates the value of the spline (spl) at every x-coordinate (x=pt[1]) of the solution points ($antw1 defined in the question algorithm) and calculates the difference to the y values expected in the solution (pt[2]). The errors of the single points are accumulated to give a total error value (“erro”).

if erro<=0.4 then grade:=grade+0.5;

end if;

If the accumulated error is smaller than 0.4, 0.5 is added to the “grade” variable

if (is(eval(diff(spl, x),x=-3), negative)) and (is(eval(diff(spl, x),x=-1), positive)) and (is(eval(diff(spl, x),x=1), positive)) and (is(eval(diff(spl, x),x=3), negative)) then grade:=grade+0.5;

end if;

More conditions that must be met to sum 0.5 to the “grade” variable.

return grade;

Exit and return the value of the grade to Maple T.A.

end if;

Ends if condition started in the 2nd line.

In practice the grading is done within the “if” clauses. Many conditions can be added to the if clause. The table below explains the functions and commands I used the most to code those conditions:

|  |  |
| --- | --- |
| command | explanation |
| is(a,prop) | evaluates if a has the property “prop” |
| is(a>b) | evaluates if a is greater than b |
| eval(f, x=a) | evaluates the function f at x=a |
| evalb(a>b) | evaluates the Boolean value (true or false) of the inequality (equivalent to is) |
| diff(f,x) | derivative of the function f with respect to x |
| abs(a) | absolute value of a |
| int(f,x=a..b) | integral of the function f between a and b |

The table below shows examples of conditions used in the grading:

|  |  |
| --- | --- |
| description | code |
| value at x=3 is negative: | (is(eval(spl,x=3),negative)) or (is(eval(spl,x=3)<0)) |
| value in x=2 is greater than 1: | (is(eval(spl,x=2)>1)) |
| increasing in x=2: | (is(eval(diff(spl, x),x=2), positive)) |
| concave in x=2: | (is(eval(diff(diff(spl,x),x),x=2),positive)) |
| maximum near x=2 or horizontal: | (evalb(abs(eval(diff(spl,x),x=2.00001))<0.05))  (writing just x=2 gives an error). 0.05 is a tolerance factor  This is equivalent to  (is(abs(eval(diff(spl,x),x=2.00001))<0.05)) |
| compare areas under curve in the intervals [0,2] and [2,4]:  (in this case we check whether they are within a factor of 2 from each other) | (is(abs(int(spl, x = 0 .. 2))/abs(int(spl, x = 2 .. 4)>2) and abs(int(spl, x = 0 ..2))/abs(int(spl, x = 2 .. 4)) > .5)) |

Another potentially useful tool is to calculate the places where a function is zero along a given interval. This is achieved with:

a:=Student[Calculus1][Roots](f, 0 .. 1);

where *f* is the function and [0,1] the interval. The result (a) is a list of x-coordinates with the positions. This can be used to find the locations of maxima and minima (places where df/dx=0):

localmaxmin:=Student[Calculus1][Roots](diff(f,x), 0 .. 1);

One must take care that the function f is defined before this command. In the grading code usually we use this command to study the spline given by the student (spl) so the command should be placed after spl is defined.

With these values one can check several things like (assuming that one used the two commands above):

* number of maxima =2: (is(nops(localmaxmin) = 2))
* first maximum before 2: (is(localmaxmin[1] < 2))
* first maximum at y=1: (is(abs(eval(spl,x=localmaxmin[1]-1)) <0.1))
* function crosses 0 before 1: (is(a[1] < 2))

Using the number of maxima as a condition must be done very carefully because when students use 3 points to define a maximum, it normally adds a barely visible maximum and the grading is incorrect. For that reason we abandoned this condition.

### Algorithm

The solution points, Math App axes grid spacing and size are defined in the question algorithm. An example of an algorithm for the “SketchApp” worksheet is given below:

$antw1=maple("[[0,0],[6,-1],[7,-1]]");

$listranges=maple("[0..10,-2..2]");

$horlabel=maple("");

$vertlabel=maple("I\_\_l/S\_\_0");

$height=400;

$width=800;

$npt=30;

$grid=10;

$BgPoints=maple("[[-10, 0],[-7, 0],[-6.5, 0], [-6, 0], [0, 1.5], [6, 0], [6.5, 0],[7, 0],[10, 0]]");

$Bgcolor="gray";

$Bgrangespline=maple("0..10");

$Bgf=maple("0");

$Bgrangef=maple("-0..0");

$Bgg=maple("0");

$Bgrangeg=maple("-0..0");

$answerpoints=maple("[[0,0],[6,-1],[6.1,-1],[7,-1],[10,-1]]");

$answerplot=plotmaple("p1 := plot(CurveFitting[Spline]($answerpoints,x), x = 0 .. 10, thickness=2,color=blue):p2 := plot($answerpoints, style = point, symbol = solidcircle, symbolsize = 20,color=brown):plots[display]({p1,p2},view=[0..10,-2..2],labels=[``,``]),axis = [gridlines = [majorlines = 1]]");

And the explanation, line by line:

$antw1=maple("[[0,0],[6,-1],[7,-1]]");

Defines the solution points (points where the spline should pass within a margin of error defined in the grading algorithm)

$listranges=maple("[0..10,-2..2]");

Defines the axes ranges

$horlabel=maple("");

$vertlabel=maple("I\_\_l/S\_\_0");

Defines the labels of the axes.

$height=400;

$width=800;

Defines the size of the Math App (I found that there’s a maximum size for both diemnsions, which I think is 800)

$npt=30;

Maximum number of points allowed for the student to place.

$grid=10;

Number of divisions for each grid point. It controls the resolution (how close can the points be)

$BgPoints=maple("[[-10, 0],[-7, 0],[-6.5, 0], [-6, 0], [0, 1.5], [6, 0], [6.5, 0],[7, 0],[10, 0]]");

$Bgcolor="gray";

$Bgrangespline=maple("0..10");

These lines define a background function that sometimes we want to show the students for guidance. The first line defines the points that create the spline, the second defines the color and the 3rd defines the range (most of the times it is the same range as the x-axis).

$Bgf=maple("0");

$Bgrangef=maple("-0..0");

$Bgg=maple("0");

$Bgrangeg=maple("-0..0");

These lines define more functions that can be added as a background. Usually we don’t use them

$answerpoints=maple("[[0,0],[6,-1],[6.1,-1],[7,-1],[10,-1]]");

$answerplot=plotmaple("p1 := plot(CurveFitting[Spline]($answerpoints,x), x = 0 .. 10, thickness=2,color=blue):p2 := plot($answerpoints, style = point, symbol = solidcircle, symbolsize = 20,color=brown):plots[display]({p1,p2},view=[0..10,-2..2],labels=[``,``]),axis = [gridlines = [majorlines = 1]]");

These two lines define a plot with an example of an answer that we show students in the feedback. The first line defines the points that build the spline and the 2nd makes the plot.

For the “SketchAppExtraGrid” worksheet, the variable $grid is replaced by

$gridx = 10;

$gridy = 4;

Represent the grid resolutions in both axis.

$factorgridx=1;

$factorgridy=10;

Represent a factor that multiplies the grid units to define the resolution.

For example, with these values, the x axis will have 10 divisions by unit and in the y-axis each division will correspond to 2.5 units.

### Feedback

In the Feedback we usually show an example of a correct answer. In the Feedback field we write:

Example of a correct answer:

$answerplot

# Yearly randomization of stage A questions

Stage A questions have a randomization that is “locked” for a year. The MapleTA questions read the randomized values from a file that is uploaded t the class file manager. This file is built using an Excel file with VBA macros that perform the randomization and call a Maple script to export the randomized values to an appropriated file format.

## The Excel file

The Excel file named “maple.xlsm” contains information about all randomized questions, i.e., the type and parameters of the randomization.

### Chapter “Sheets”

For example the table correspondent to Chapter 2 is shown in Figure 9. There’s one worksheet like this for each chapter.

Each line represents a question or a particular randomization in a question. I chose to have all questions represented, even the ones that are not randomized. Questions that have more than one randomized variables occupy more than one line.

The column A is the question name and it is there only for reference. It doesn’t play any role in the process.

The column B is the type of randomization. It tells the VBA script which kind of randomization is to be done in this line. Available randomizations are:

* rint: produces a random integer in column F lower than the value present in column C (non-negative)
* vector: shuffle the numbers present in columns F and further until an empty cell.
* range: same as rint but value in column F is lower **or equal** than column C

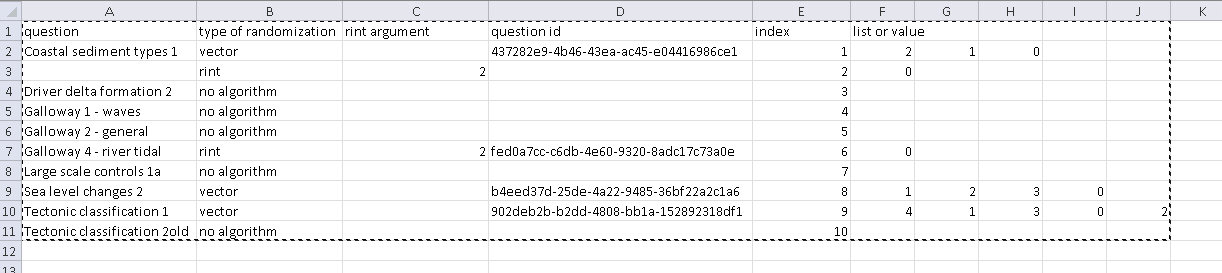


Figure 9 – table with randomizations in Chapter 2

Column D has the question ID’s string. This string is the key for the algorithm on Maple TA to find the right value. It must match the $id variable on the question algorithm[[2]](#footnote-2).

Column E is just an index for each question and or randomization. It tells the VBA script that this line should be considered. It is important that all randomizations have an index.

Columns F and further hold the randomized values.

### Export “Sheet”

This Sheet has the values that are actually exported. The randomized values, question ids and indexes of all Chapter Sheets are copied into one table here.

### Start “Sheet”

The Start Sheet has the button that starts the script that randomizes the values and the button that creates the files that have to be uploaded to the Maple TA server.

As can be read in the Start Sheet, there are two important conditions for these scripts to work:

1. the file libexcel.mpl must be in the same directory as this workbook. This file contains the Maple script necessary to export the values into .lib and .ind files which can be read by Maple TA-
2. The path to the maple.exe file on cell G11 must be correct. Maple must be installed on the computer.

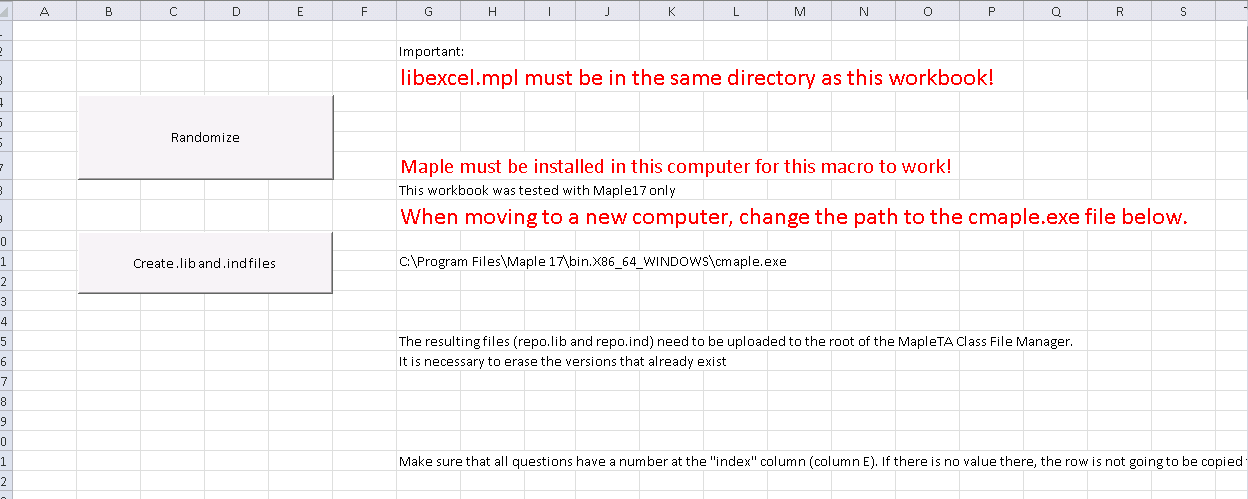


Figure 10 – Start “sheet”

If everything goes well the files repo.lib and repo.ind will be created in the same directory of the workbook. These must be uploaded to the class file manager. The previous existing files with the same name must be deleted before.

## Maple TA algorithm

In order to read the variables from the repo.lib and repo.ind files the questions must have the right code. An example is shown below:

$notused="$A=maple("randomize():StringTools:-Randomize():combinat[randperm]([0,1,2])");";

$notused="$choice=rint(2);";

$id="437282e9-4b46-43ea-ac45-e04416986ce1";

$d=maple('ListTools:-Search("$id", A[1..-1,1]), libname=/tudelft/web/Mathcla000/Public\_Html/repo.lib');

$A=maple("convert(A[$d,3..5],list), libname=/tudelft/web/Mathcla000/Public\_Html/repo.lib");

$choice=maple("A[$d+1,3], libname=/tudelft/web/Mathcla000/Public\_Html/repo.lib");

The §notused variables are not used (who would have guessed!). They are there to keep the old randomization code (there’s no such thing as a comment in Maple TA algorithms).

The important lines are:

$id="437282e9-4b46-43ea-ac45-e04416986ce1";

Defines the question id. This string must match the one in the Excel file (see Figure 9 where the id for the Coastal Sediment types 1 is the same as above).

$d=maple('ListTools:-Search("$id", A[1..-1,1]), libname=/tudelft/web/Mathcla000/Public\_Html/repo.lib');

This line searches the repo.lib for the string ($id) defined in the line above. It returns the line index of the localization ($d)

$A=maple("convert(A[$d,3..5],list), libname=/tudelft/web/Mathcla000/Public\_Html/repo.lib");

Reads what is in the columns 3 to 5 of the line with index equal to $d. In the Excel file these correspond to columns G to F. It is necessary to set the end column manually. Here it is 3 to 5 because there are 3 values. On line 9 (question Sea level changes 2) there are 4 values and thus the code must change to: …A[$d,3..6]…

$choice=maple("A[$d+1,3], libname=/tudelft/web/Mathcla000/Public\_Html/repo.lib");

Reads the value in column 3 of the next line ($d+1). Notice that the rint randomization is in the second line in Figure 9.

1. The switch function operates with zero being the first index. If index=0 $term1 would be “Groynes”. [↑](#footnote-ref-1)
2. In reality question ID strings changed when the question bank migrated to a different server. However, this doesn’t affect the process because the algorithm keeps the old value. [↑](#footnote-ref-2)