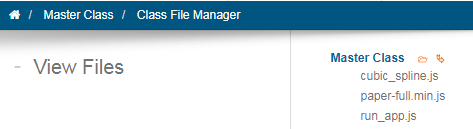
# HTML Questions

HTML question types are questions that take use JavaScript and HTML for the definition of the interface with the student. The input of the students to this application is recorded and passed on to MapleTA. Grading is done by the Maple engine in MapleTA. The only HTML question now we use is a Sketch question developed by Anatoly Ilin.

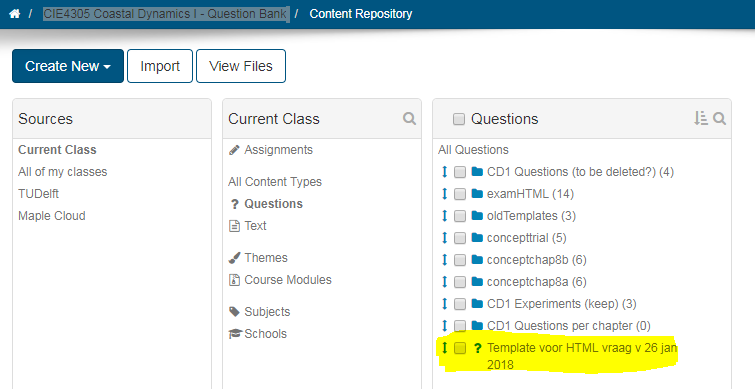
# HTML – Sketch

In this type of question, students draw a line by placing points on a Cartesian axis. The sketching functionality is defined in the HTML . You can see the code if you open the HTML example question / template in MapleTA. This specific question type for Coastal Dynamics requires a 3 files which are uploaded in the Master Class on the MapleTA server. You can contact the functional admin for information on that specific file.



One doesn’t need to change the HTML code in order to use it in Maple TA. It’s parameters (e.g. axes ranges, labels, etc.) are accessible from the algorithm dialog in the question designer. In this section I present the code for the grading and for the algorithm.

To create a new HTML question you should start by cloning the (most recent) template that is present in the root group of **CIE4305 Coastal Dynamics I - Question Bank**



Algorithm

|  |  |  |
| --- | --- | --- |
| Parameter | notation | explanation |
| $teachermode = | "false"; or “true”; |  |
| $antw = | "[[-6,0],[-2,0],[-1,0.5],[0,1.5],[1,0.5],[2,0],[6,0]]"; | These are the points that get checked for the 'erro' parameter in the grading. Most times some crucial points get checked, not all points that need to be drawn by the student. |
| $antw1= | maple(“$antw”); | Necessary notation for grading code |
| $anspoints= | "[[-6,0],[-2,0],[-1,0.5],[0,1.5],[1,0.5],[2,0],[6,0]]"; | This generates the plot for the correct answer - it should display the correct curve. |
| $answerplot= | plotmaple("p1 := plot(CurveFitting[Spline]($anspoints,x), x = 0 .. 6, thickness=2,color=blue):p2 := plot($answerpoints, style = point, symbol = solidcircle, symbolsize = 20,color=brown):plots[display]({p1,p2},view=[0..6,-3..3],labels=[``,`I\_\_l/S\_\_0`]),axis = [gridlines = [majorlines = 1]]"); | Not necessary |
| $axes = | "[-10,10,-2,2]"; | This list defines the x-axis [-10,10] and y-axis [-2,2]  *In mathapp questions this is $listranges.* |
| $canvasWidth =  $canvasHeight = | "800px"; "400px"; | Fixed values for correct layout in different views |
| $canvasDef = | "{ width: 800,  height: 400,  vPad: 20,  hPad: 20 }"; | This defines the size of the canvas. Max width should be 800.  Enlarge values for padding (vPad and hPad) when axis labels and axis names are out of view |
| $backgroundlines = | "{ lijn1: {  x: [-10,-7,-6.5, -6, 0, 6 , 6.5, 7, 10],  y: [ 0, 0, 0, 0, 1.5, 0 , 0 , 0, 0],  lineColor: 'green',  lineColorGreyShade: -1,  lineThickness: 1} }"; | Definition of the background line. If you want to enter the coordinates of the line instead of x: [x1, x2,..xn] y: [y1,y2,..yn]] then you insert  **bgcoord: [[x1,y1],[x2,y2]..[xn,yn]],**  insert this above”x:” and you can leave the definition of x: and y: as is  if the background line extends beyond the defined grid, add  x\_limit\_min: xmin,  x\_limit\_max: xmax |
| $axis\_defintion = | y\_axis\_position: 'auto',  y**LabelNumberPrecision**: 0,  yLabelColor: 'black',  yLabelJustification: 'center',  y**LabelPositionVertical**: +5,  y**LabelPositionHorizontal**: 15,  yLabelShowZero: false,  y**AxisFlipped**: false,  yAxisArrow: true,  yFontSize: 10,  y**AxisName**: 'S/Snet' ,  yAxisNameFontSize : 15,  yAxisNameFontColor : 'black',  yAxisNameJustification : 'center',  y**AxisNameVertical**: 0,  y**AxisNameHorizontal**: 15,  y**AxisNameOrientation**: 90,  AxisLineColor: 'black',  AxisLineThickness: 1,  AxisArrowSize: 10,  AxisArrowAngle: 135,  AxisArrowLineThickness: 1,  AxisArrowLineColor: 'black' | Only the relevant properties are ‘explained’ x and y have the same parameters   * **LabelNumber precision** needs to be 1 if you have decimals in the axis label. * Adjust **Label Position** if you want to move the axis labels * Is some situations the axis needs to be flipped. * Defines the **unit** on the axis (no formatting possible) * Defines the **positioning of the AxisName** . |
| $minor\_grid\_lines | **xStep**: 0.1,  **yStep**: 0.1,  lineWidth: 0.2,  lineColor: 0.5,  checkmark\_offset: 3,  checkmark\_color: 'grey',  checkmark\_width: 0.5 | If necessary you can adjust the step size of the minor gridlines.  If changing the step size causes your axis labels to ‘mess up’, try adjusting the  **LabelNumber precision** in $axis-definition**.** |
| $major\_grid\_lines = | xStep: 1,  yStep: 1, etc. | See minor grid lines |

## Grading code

The grading code has to be entered here:

The basic code in the template is:

curv:= $RESPONSE[1]:

grade := 0: erro:= 0:

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

for pt in $antw1 do erro:=erro+abs(pt[2]-eval(spl,x=pt[1])): end do:

if erro<=0.4 and is(abs(eval(diff(spl,x),x=0))<0.1) then grade:=1;

end if: grade;

The grading code snippets Joao has defined can be used in the HTML question grading code.

**curv:= $RESPONSE[1]:**

Captures the points that describe the spline that is created in HTML, $RESPONSE[3]: captures the points the student placed in the grid.

**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.

**grade;**

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

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):

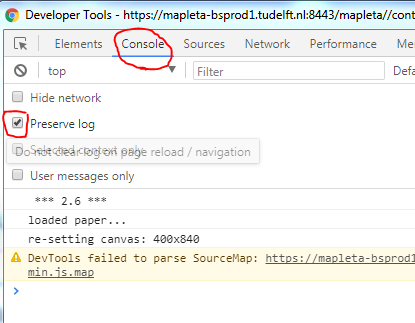
* number of maxima =2: (is(nops(localmaxmin) = 2))
* first maximum before 2: (is(localmaxmin[1] < 2))
* first maximum at y=1: (is(abs(eval(spl,x=localmaxmin[1]-1)) <0.1))
* function crosses 0 before 1: (is(a[1] < 2))

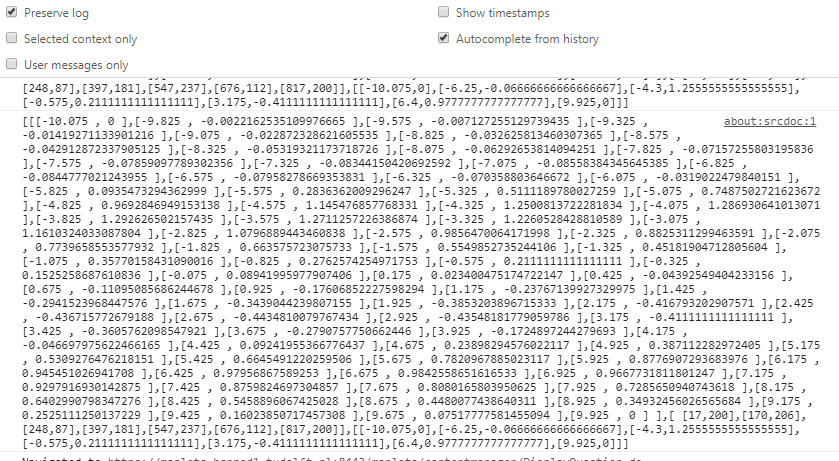
Using the number of maxima as a condition must be done very carefully because when students use 3 points to define a maximum, it normally adds a barely visible maximum and the grading is incorrect. For that reason we abandoned this condition.

To create new questions in MapleTA, you will want to use Maple to check your code and find the best x-values to check your grading.

There is a Maple-sheet named ‘Manual Maple Sheet-curves’ that provides the basic code.

To capture the data-points from your question-preview take the following steps:

1. Open the preview of your question and right click that window to select ‘ Inspect’.
2. In the new window (Developer Tools) open the second tab ‘Console’.
3. To make sure that Chrome will not delete the log when navigating to the grading, check the ‘preserve log’ box.  
   
4. Now draw the graph a student would make and ‘grade’ the response. The data will be captured by the console.



The format of these data points is:

Format [ [Cubic monotonic Spline punten, HTML5 opgave] , [ (student) punten in browser / viewport coördinaten ] , [ (student) punten in x/y stelsel coördinaten ] ]

Data[1] geeft de Cubic monotonic Spline punten (by HTML)  
Data[2] geeft de (student) punten in browser / viewport coördinaten   
Data[3] geeft de (student) punten in x/y stelsel coördinaten