SketchApp – The Möbius interactive spline drawing tool

# Introduction

[Möbius](https://www.digitaled.com/mobius/) is the online STEM learning platform from [DigitalEd](https://www.digitaled.com/). One of its features is the [HTML question type](https://www.digitaled.com/support/help/instructor/Content/INST-AUTHORING/QUESTION-TYPES/Author-HTML-question.htm) that can be used for as a *response area*. When inserting a HTML response area into a question, an HTML <iframe> element is created. An iframe is simply used to embed a HTML page in the current one. Using an iframe ensures that the parent page is not affected.

## Setting up a HTML response area

When editing a HTML response area you are prompted with the following options:

Table 1: HTML response area edit options

|  |  |  |
| --- | --- | --- |
|  | **Weighting** | The weighting of the response area (any integer greater than 0). This is proportional to the question total.  Default = 1 |
| **Answer** | The correct response, referenced in the Grading Code as $ANSWER.  *Note: For the Sketchapp we use “$answer”.* |
| **Grading Code** | Used to evaluate the student response ($RESPONSE) with the correct answer ($ANSWER). The code used must use valid Maple code (and syntax). |
|  | **Question HTML** | Here you can define HTML code needed to display what you want (such as <divs> and <script> elements, etc.). Note that this ONLY applies to the inserted <iframe>. |
| **Question CSS** | Here you can define CSS code to change how your HTML code looks. Note that this ONLY applies to the inserted <iframe>.  *Note: For the SketchApp this is left empty.* |
| **Question JavaScript** | Question JavaScript requires three functions to work: *initialize(interactiveMode)*, *setFeedback(response,answer)* and *getResponse()*. They are explained below. |

Definitions of the below functions are quoted from DigitalEd Support:

* initialize(interactiveMode) Called whenever the response area is displayed to either prompt the student for a response or show the student’s response and the correct answer.
* setFeedback(response,answer) Called when the student’s response and the correct answer are to be displayed next to each other (the variable response will receive the output of *getResponse()* and the variable answer will receive the value defined in the **Answer** field).
* getResponse() Called whenever the question is graded and returns the state of the response are such that it can be evaluated by the grading code.

If you are using variables ($VarName) in the **Algorithm** field of the question and want to use them for your HTML response area, you can reference them by writing the following code in **Question JavaScript**:

var VarName = $VarName;

# SketchApp overview

The SketchApp is built within the inserted iframe and requires three JavaScript scripts:

* RunApp.js This is the actual code that creates the SketchApp
* Paper-full.js [Paper.js](http://paperjs.org/) is used to enable interaction with the canvas
* Cubic\_spline.js This script enables Cubic Monotone (Hermite) spline interpolation

The SketchApp is designed to draw interactive splines on top of a graph with some buttons. An example is given in Figure 1.

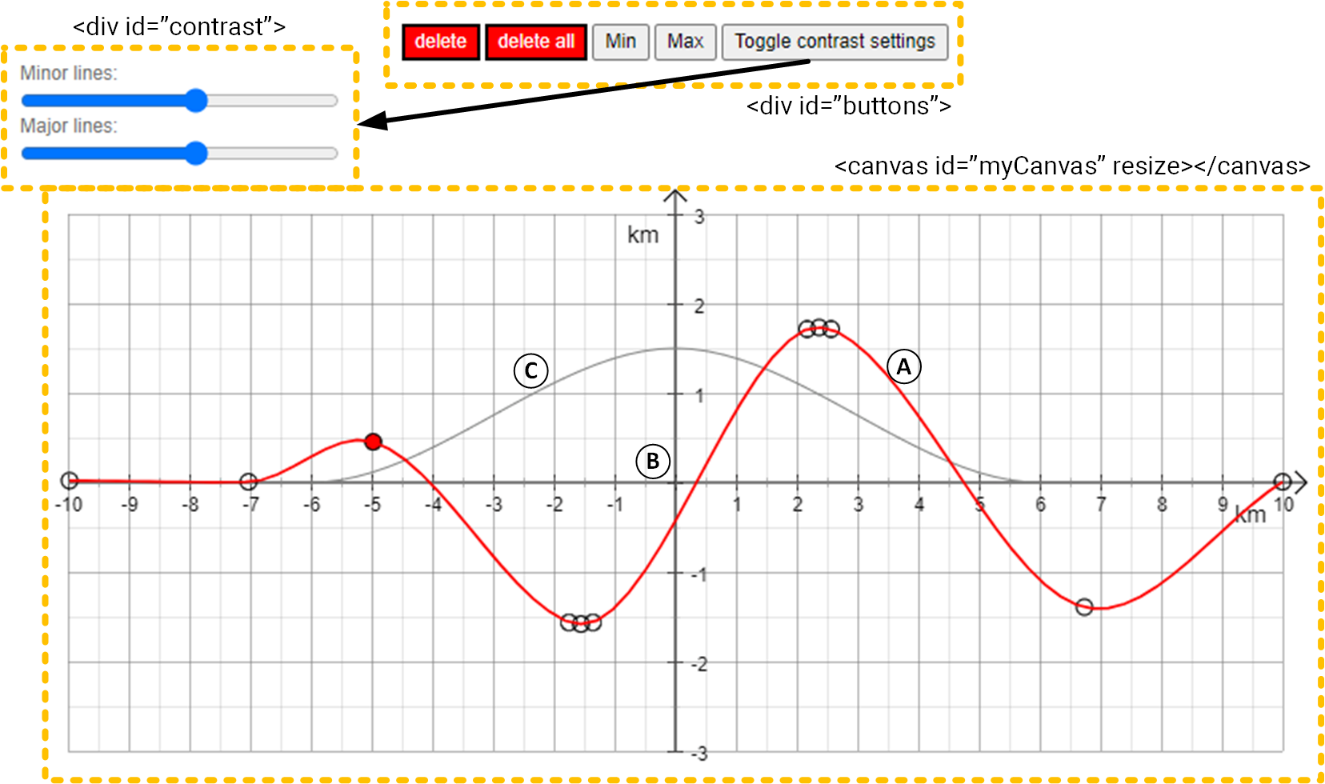


Figure 1: SketchApp example. (A) Spline (B) Background consisting of axes with labels and a grid (C) Spline as background

## Setting up the iframe

The first step is to create a <div id=”buttons”> and <canvas id=”myCanvas”>[[1]](#footnote-1) element in the **Question HTML** part of the response area (Table 1, Figure 2). In the <div> element, we define the necessary HTML buttons using <input type=”button”> elements. A second, nested, <div id=”contrast”> element is created in which we store two sliders using <input type=”range”> elements. The sliders are hidden by default and can be toggled using the *toggleContrast* button. Button functionality is coded in RunApp.js. In Figure 2 the full **Question HTML** code is provided.

<div id="buttons" style="text-align:center">

<input type="button" id="delPoint" value="delete" style="color: white; background-color: red;" />

<input type="button" id="delAll" value="delete all" style="color: white; background-color: red;" />

<input type="button" id="buttonMin" value="Min" style="color: black;" />

<input type="button" id="buttonMax" value="Max" style="color: black;" />

<input type="button" id="toggleContrast" value="Toggle contrast settings" style="color: black;" />

<div id="contrast" style="display: none; text-align:left">

Major lines: <input type="range" id="gridMajor" style="width: 200px;" min="0" max="9" value=""+major\_grid\_lines.lineColor\*10>

Minor lines: <input type="range" id="gridMinor" style="width: 200px;" min="0" max="9" value=""+minor\_grid\_lines.lineColor\*10>

</div>

</div>

<canvas id="myCanvas" resize></canvas>

Figure 2: Möbius HTML response area - Question HTML part

## Defining Algorithm variables

The SketchApp uses variables defined in the **Algorithm** field to draw the background graph. Also, here we define the correct answer ($answer) for the **Answer** field. We define the correct answer here instead of directly in the **Answer** field because the SketchApp also needs this to draw the correct answer in the gradebook. The short list is given below, the full list is provided in Appendix A. Red text is user adjustable.

Table 2: Short list of Algorithm field variables

|  |  |
| --- | --- |
| $teachermode = "boolean"; | If true, the coordinates of the drawn points are displayed in a <div> element below the SketchApp (but still in the <iframe>). |
| $answer = "[[x1,y1],[x2,y2],…,[xn,yn]]"; | The correct answer of the question (floats). |
| $axes = "[xmin,xmax,ymin,ymax]"; | Set the graph axis limits (floats). |
| $answerplot = plotmaple("…"); | This displays the correct answer without the need to preview the question. Updates when *save* is clicked. |
| $canvasDef = "{…}"; | Set canvas width and height and vertical and horizontal padding. |
| $backgroundlines = "{…}"; | If you want to draw a spline in the background, define it here. |
| $axis\_definition = "{…}"; | Here all the variables for the axes are defined. |
| $major\_grid\_lines = "{…}"; | Define where the gridlines are drawn and what their style is. |
| $minor\_grid\_lines = "{…}"; |
| $interaction\_settings = "{…}" | Min/Max button functionality and spline drawing settings. |

## Setting up the Question JavaScript

The **Question JavaScript** field contains the HTML response area required functions mentioned earlier, the Möbius variables ($VarName) as JavaScript variables (var VarName) and some helping variables are defined. The full list is provided below. Comments and console.log() events are omitted. Red text is user specific.

Table 3: Full list of Question Javascript field

|  |  |
| --- | --- |
| var teachermode = $teachermode;  var axes = $axes;  var canvasDef = $canvasDef;  var axis\_definition = $axis\_definition;  var major\_grid\_lines = $major\_grid\_lines;  var minor\_grid\_lines = $minor\_grid\_lines;  var backgroundlines = $backgroundlines;  var interaction\_settings = $interaction\_settings;  var type = 0;  var AnswerStr = "";  var errormessages ="";  var gradebook = false; | * type defines the behaviour of RunApp.js * Answerstr is used for grading. * errormessages is used when teachermode is true to display errors when editing a SketchApp question. |
| jQuery.getScript('/web/Cie4305000/Public\_Html/.../RunApp.js', function(){}); | Define the path to the RunApp.js. Here Cie4305000 is for the child class. To use the same path across all classes (or servers) the file should be placed in Masterclass, which is only accessible by Möbius administrators. |
| function initialize(interactiveMode) {  gradebook = !interactiveMode  if (gradebook){  jQuery( "#buttons" ).remove();  }  }; | When the question is graded, *interactiveMode* is false and we disable interaction. Thus, we have to remove the <div id=”buttons”> element. |
| var translations = ["No answer", "Aucune réponse", "Keine Antwort", "Nessuna risposta", "解答なし", "未解答", "Sin respuesta", "Geen antwoord", "Καμία απάντηση", "답변 없음", "Brak odpowiedzi", "Sem resposta"]; | This is a helping variable for the RunApp.js behaviour, see *function setFeedback()* below. |
| function setFeedback(response, answer) {  if (translations.indexOf(response) >= 0 ) {  type = 1;  runApp(response, type);  }  else if (answer == null) {  type = 2;  runApp(response, type);  }  else if (answer != null) {  type = 3;  runApp(answer, type);  }  }; | *response* is either “No answer” (or translation) or student response.  *answer* is either *null* or correct answer.   * type = 1: Check if *response* is found in *translations* array. If true, question is opened for the first time or student has not provided a response yet. Thus, allow interaction (mouse and buttons). * type = 2: Show student response. Load student response. If ungraded, allow interaction (mouse and buttons). If graded, disable interaction. * type = 3: Show correct answer. Load correct answer but do not display the points and disable interaction. |
| function getResponse() {  return AnswerStr;  }; | RunApp.js builds the AnswerStr. Here, AnswerStr is returned by the function as $RESPONSE, which in turn can be used in the **Grading Code** field. |

# Building the SketchApp

## Main function

### Initialisation

The full code is contained in one function, runApp(array, type). The required scripts (*paper-full.js* & *cubic\_spline\_original.js*) are called first.

function runApp(*array*, *type*) {

    jQuery.getScript('path/to/cubic\_spline\_original.js', function() {

        jQuery.getScript('path/to/paper-full.js', function() {

.

.

.

       };

   };

}

Then, a PaperScope and Tool object is created to access the Paper.js classes. Next, the canvas is given an initial size defined by the canvasDef variable and an empty project is setup:

var scope = new paper.PaperScope();

var tool = new scope.Tool();

$("#myCanvas").width( canvasDef.width );

$("#myCanvas").height( canvasDef.height );

scope.setup($("#myCanvas")[0]);

When teachermode is True, a <div id=”teacher”> element is appended. Later in the code a HTML table is created in this <div> element that tracks the points created by the user and show any error messages prompted by the code:

if(teachermode) {

var teacherDiv = document.createElement('div');

    teacherDiv.id = 'teacher';

    teacherDiv.className = 'teacher';

    teacherDiv.style="overflow-y:scroll; height:300px";

    document.getElementsByTagName('body')[0].appendChild(teacherDiv);

}

The necessary “global” variables are defined, of which only three are discussed here: PermanentElements, DrawnPoints and SplinePoints. All three are a Group object from Paper.js that stores information to draw. PermanentElements contains everything to draw the background graph (axes, gridlines, labels and an optional background spline). DrawnPoints contains the Shape.Circle(), which are drawn to show where the user has created a point. SplinePoints also contains the coordinates of these points, and the interpolated points to draw the spline.

var PermanentElements = new scope.Group();

var DrawnPoints = new scope.Group();

var SplinePoints = new scope.Group();

.

.

.

Lastly, we call the draw\_axis() function to draw the background graph. This function is explained in section 3.2.

draw\_axis();

Now we are ready to interact with the canvas depending on what behaviour (i.e., *type*) the App is running in. The functions buttons(), interact() and draw\_spline() define the behaviour:

* buttons() Enables the HTML button functions, see section 3.3.
* interact() Allow interaction with the canvas, see section 3.4
* draw\_spline() Draw the interpolated spline, see section 3.5.

### Behaviour

Type 1 behaviour occurs when the question is initialized and the student has not yet given a response, thus enabling button and mouse interaction. Type 2 behaviour occurs when the question is loaded and the student has given a response. Two sub behaviours are possible, a graded or ungraded question. For a graded question (gradebook is called), the student answer is simply loaded and drawn. For an ungraded question, the student answer is loaded, drawn and interaction (button & mouse) is enabled. Type 3 occurs when the gradebook is called and draws the correct answer spline while disabling button and mouse interaction.

if (*type*==1) {

    buttons();

    interact();

}

else if (*type*==2) {

.

.

.

    if (gradebook) {

.

.

.

        draw\_spline();

    }

else {

.

.

.

    buttons();

    interact();

    draw\_spline();

    }

}

else if (*type*==3) {

.

.

.

    draw\_spline();

}

else {

console.log("Error! incorrect type for RunApp(array,type). Type is 1, 2 or 3.");

}

## Building the background graph

### Helper functions and canvas resizing

The background graph contains the axes, gridlines, labels and an optionally drawn spline. Drawing the background is done by calling the function draw\_axis(). The function contains four helping functions:

* Draw\_line(...) Draws a line. Used to draw the major and minor gridlines, and major and minor checkmarks.
* Draw\_arrow(...) Draws a line with an arrow tip. Used for the axes.
* Draw\_label\_text(...) Draws the text for the axis labels (not the axis name).
* Even\_fix(...) A fix for Chrome and Edge displaying gridlines.

The code for the above functions, including a more detailed explanation can be found in Appendix B.

Before building the background, everything from PermanentElements is deleted first:

PermanentElements.removeChildren();

To properly draw the graph the entire graph should fit the canvas, which is not guaranteed by default. We explain this using an example:

* A canvas with width = 600 pixels and hPad = 30 pixels. The graph is thus 540 pixels wide.
* X-axis is from 0 to 7.
* Major gridline for the X-axis is 1, minor is 0.2.
* The pixel distance for major gridlines is then 77.14 pixels, minor is then 15.43 pixels.
* Pixels are always integers, and thus the graph will be incorrectly drawn if we round these numbers to 77 and 15 pixels.

To avoid this issue, the canvas is resized in width and height using minor\_grid\_lines.xStep and yStep. The minor gridline step in pixels is calculated and rounded to an integer. Then, the new canvas width and height are calculated and the canvas is resized. Calling scope.viewSize rescales the Paper.js canvas. Lastly, the major gridlines pixel step is calculated.

minorX\_BrowserStep = Math.round((canvasDef.width  - canvasDef.hPad\*2) / ((Math.abs(axes[1] - axes[0]) / minor\_grid\_lines.xStep)));

minorY\_BrowserStep = Math.round((canvasDef.height - canvasDef.vPad\*2) / ((Math.abs(axes[3] - axes[2]) / minor\_grid\_lines.yStep)));

canvasDef.width  = minorX\_BrowserStep \* ((Math.abs(axes[1] - axes[0]) / minor\_grid\_lines.xStep)) + canvasDef.hPad\*2;

canvasDef.height = minorY\_BrowserStep \* ((Math.abs(axes[3] - axes[2]) / minor\_grid\_lines.yStep)) + canvasDef.vPad\*2;

$("#myCanvas").width( canvasDef.width );

$("#myCanvas").height( canvasDef.height );

scope.viewSize = [canvasDef.width, canvasDef.height];

majorX\_BrowserStep = Math.round((canvasDef.width  - canvasDef.hPad\*2) / ((Math.abs(axes[1] - axes[0]) / major\_grid\_lines.xStep)));

majorY\_BrowserStep = Math.round((canvasDef.height - canvasDef.vPad\*2) / ((Math.abs(axes[3] - axes[2]) / major\_grid\_lines.yStep)));

### Determine axes locations

The next step is to determine the browser coordinates of the X- and Y-axis. As three user options are available (auto, left & right for the Y-axis OR top & bottom for the X-axis) we need to address this in the code. Below a minimal version is shown, the full code can be found in Appendix C.

**X-axis position**

if (axis\_definition.y\_axis\_position != "auto" ) {

if (axis\_definition.y\_axis\_position == "left") {

    y\_axis\_x\_coordinate = canvasDef.hPad;

    }

    else if (axis\_definition.y\_axis\_position == "right") {

    y\_axis\_x\_coordinate = canvasDef.width - canvasDef.hPad;

    }

    else {

    console.log("y\_axis\_position undefined, selector: " + axis\_definition.y\_axis\_position + " unknown. [auto, left, right]");

    }

}

*/\* determine location when option is 'auto' \*/*

else {

.

.

.

}

**Y-axis position**

if (axis\_definition.x\_axis\_position != "auto" ) {

if (axis\_definition.x\_axis\_position == "top") {

    x\_axis\_y\_coordinate = canvasDef.vPad;

    }

    else if (axis\_definition.x\_axis\_position == "bottom") {

    x\_axis\_y\_coordinate = canvasDef.height - canvasDef.vPad;

    }

    else {

    console.log("x\_axis\_position undefined, selector: " + axis\_definition.x\_axis\_position + " unknown. [auto, top, bottom]");

    }

}

*/\* determine location when option is 'auto' \*/*

else {

.

.

.

}

### Drawing the axes, labels and gridlines

Now that the canvas has the correct size for the gridlines and know the axes locations we can start drawing the background graph. First the X-axis, major vertical gridlines and check marks and X-axis labels are drawn. Then the Y-axis, major horizontal gridlines and checkmarks and Y-axis labels are drawn. To finish the gridlines, the minor vertical and horizontal gridlines and check marks are drawn. Lastly, the axis names are drawn. A minimal version of the code is shown below. The full code can be found in Appendix D.

**X-axis, major vertical gridlines and check marks & X-axis labels**

if (!axis\_definition.xAxisFlipped) {

*/\* Draw x\_axis, not flipped. Extend line 0.5\*hPad beyond graph area \*/*

    if (axis\_definition.xAxisArrow) {

        draw\_arrow(...));

    }

    var x\_temp = even\_fix(canvasDef.hPad, 0.5);

}

else {

*/\* Draw x\_axis, flipped. Extend line 0.5\*hPad beyond graph area \*/*

    if (axis\_definition.xAxisArrow) {

        draw\_arrow(...));

    }

    var x\_temp = even\_fix(canvasDef.width - canvasDef.hPad, -0.5);

}

for (var i = axes[0]; i <= axes[1] ; i = i + major\_grid\_lines.xStep) {

*/\* Draw major vertical grid line \*/*

    draw\_line(...);

*/\* Draw major vertical checkmark \*/*

    draw\_line(...);

*/\* Draw x\_axis labels \*/*

    draw\_label\_text(...);

    if (!axis\_definition.xAxisFlipped) {

        x\_temp = (x\_temp + majorX\_BrowserStep);

    }

    else {

        x\_temp = (x\_temp - majorX\_BrowserStep);

    }

}

**Y-axis, major horizontal gridlines and check marks & Y-axis labels**

if (!axis\_definition.yAxisFlipped) {

*/\* Draw y\_axis, not flipped. Extend line 0.5\*vPad beyond graph area \*/*

    if (axis\_definition.yAxisArrow) {

        draw\_arrow(...));

    }

    var y\_temp = even\_fix(canvasDef.height - canvasDef.vPad, 0.5);

}

else {

*/\* Draw y\_axis, flipped. Extend line 0.5\*vPad beyond graph area \*/*

    if (axis\_definition.yAxisArrow){

        draw\_arrow(...));

    }

    var y\_temp = even\_fix(canvasDef.vPad, 0.5);

}

for (var i = axes[2]; i <= axes[3] ; i = i + major\_grid\_lines.yStep) {

*/\* Draw major horizontal grid line \*/*

    draw\_line(...);

*/\* Draw major horizontal checkmark \*/*

    draw\_line(...);

*/\* Draw y\_axis labels \*/*

    draw\_label\_text(...);

    if (!axis\_definition.yAxisFlipped) {

        y\_temp = y\_temp - majorY\_BrowserStep;

    }

    else {

        y\_temp = y\_temp + majorY\_BrowserStep;

    }

}

**Minor vertical gridlines and check marks & minor horizontal gridlines and check marks**

x\_temp = even\_fix(canvasDef.hPad, 0.5);

for (var i = axes[0]; i <= axes[1] ; i = i + minor\_grid\_lines.xStep) {

*/\* Draw minor vertical grid line \*/*

    draw\_line(...);

*/\* Draw minor vertical checkmark (along x-axis) \*/*

    draw\_line(...);

    x\_temp = x\_temp + minorX\_BrowserStep;

}

y\_temp = even\_fix(canvasDef.height - canvasDef.vPad, 0.5);

for (var i = axes[2]; i <= axes[3] ; i = i + minor\_grid\_lines.yStep) {

*/\* Draw minor horizontal grid line \*/*

    draw\_line(...);

*/\* Draw minor horizontal checkmark (along y-axis) \*/*

    draw\_line(...);

    y\_temp = y\_temp - minorY\_BrowserStep;

}

**X- and Y-axis names**

if (axis\_definition.xAxisName != "") {

    var text = new scope.PointText(...));

    text.justification = axis\_definition.xAxisNameJustification;

    text.fillColor = axis\_definition.xAxisNameFontColor;

    text.fontSize = axis\_definition.xAxisNameFontSize;

    text.content = axis\_definition.xAxisName;

    PermanentElements.addChild(text);

}

if (axis\_definition.yAxisName != "") {

    var text = new scope.PointText(...));

    text.rotate(axis\_definition.yAxisNameOrientation);

    text.justification = axis\_definition.yAxisNameJustification;

    text.fillColor = axis\_definition.yAxisNameFontColor;

    text.fontSize = axis\_definition.yAxisNameFontSize;

    text.content =  axis\_definition.yAxisName;

    PermanentElements.addChild(text);

}

### Finishing up

Optionally a background (sp)line can be drawn. The code for this is discussed in Appendix E. So far we have added all the Paper.js objects to the group PermanentElements. To truly draw these objects to the canvas we add PermanentElements to the active layer of the scope:

scope.project.activeLayer.addChild(PermanentElements);

## Mouse interaction

Interaction is possible in two ways, by clicking and dragging the mouse. When the user clicks, the tool.onMouseDown event is called. When the user holds the mouse button down and drags, the tool.onMouseDrag event is called. Here only the concept of what happens when either event is called is discussed. The full code for mouse interaction can be found in Appendix F.

### Clicking the mouse button

The point where the user has clicked is defined using a Point() object. The coordinates of the object are tested for a hit with the DrawnPoints group. If true, the user clicked on a drawn circle to select this point, thus only the coordinates of the click are stored. If false, it means the user wants to add a new point. First we check if any points exists in PointsLocation (a JavaScript array of the Point() objects created by clicking). If no points exist, the coordinates are appended to PointsLocation. If points already exist the new point is spliced into the array. Lastly, we call the draw\_spline() function to redraw the spline.

### Dragging the mouse

When the mouse is dragged when clicked, the starting position and distance (delta) of the mouse are stored each “system” update. If the dragged point gets close to an already existing point, defined by a tolerance, the non-dragged point is removed from PointsLocation. Each “system” update the current coordinates of the mouse are used to renew the Point() object in PointsLocation.

## Button functionality

The <input type=”button”> elements are connected with JavaScript code using jQuery, see below. The code of all the buttons can be found in Appendix G.

var delPoint = $('#delPoint'); */\* Delete the selected point \*/*

var delAll = $('#delAll'); */\* Delete all points \*/*

var buttonMin = $('#buttonMin'); */\* Force the selected point to be a local minimum \*/*

var buttonMax = $('#buttonMax'); */\* Force the selected point to be a local maximum \*/*

var toggleContrast = $('#toggleContrast'); */\* Toggle sliders to adjust gridlines \*/*

var gridMajor = $('#gridMajor'); */\* Change major gridline greyscale \*/*

var gridMinor = $('#gridMinor'); */\* Change minor gridline greyscale \*/*

### Buttons “Delete” and “Delete all”

These buttons modify PointsLocation, FittedSplineBrowserX and FittedSplineBrowserY. The “Delete” button deletes the selected point and redraws the spline. The “Delete all” button removes all points, but also removes all content from DrawnPoints and SplinePoints.

### Buttons “Min” and “Max”

These buttons modify PointsLocation, FittedSplineBrowserX and FittedSplineBrowserY. The buttons insert two more points to the three arrays. The coordinates of the new Point() objects get an offset from the selected point coordinates, defined by interaction\_settings.deltax and interaction\_settings.deltay.

### Contrast buttons

When the “Toggle contrast settings” button is clicked, the style.display attribute of the <div id=”contrast”> element is changed. Initially the value style.display = ‘none’ and changed to ‘block’ when clicked, which shows the div element in the webpage. If clicked again, the value is set back to ‘none’.

Inside the <div id=”contrast”> element two sliders are present. The sliders change the value of major\_grid\_line.lineColor and minor\_grid\_line.lineColor. In order to let these changes have an effect, the background graph is redrawn by calling draw\_axis().

## Drawing the spline

Appendix A – Algorithm field

Below the full **Algorithm** field is given. Text in red should be replaced with your own value.

|  |  |
| --- | --- |
| $teachermode = "boolean"; | If true, the coordinates of the drawn points are displayed in a <div> element below the SketchApp (but still in the <iframe>) |
| $answer = "[[x1,y1],[x2,y2],…,[xn,yn]]"; | The correct answer of the question (floats) |
| $axes = "[xmin,xmax,ymin,ymax]"; | Set the graph axis limits (floats) |
| $answerplot = plotmaple("p1 := plot(CurveFitting[Spline]($answer,x), x = $axes[1]..$axes[2], thickness=2, color=blue): p2 := plot($answer, style = point, symbol = solidcircle, symbolsize = 20, color=brown): plots[display]({p1,p2}, view=[$axes[1]..$axes[2],$axes[3]..$axes[4]], labels=[``,``])"); | This displays the correct answer without the need to preview the question. Updates when *save* is clicked. |
| $canvasDef = "{  width: integer,  height: integer,  vPad: integer,  hPad: integer  }"; | Set canvas width and height and vertical and horizontal padding (vPad and hPad, respectively) |
| $backgroundlines = "{  lijn1: {  x: [x1,x2,…,xn],  y: [y1,y2,…,yn],  lineColor: string,  lineColorGreyShade: integer/float,  lineThickness: integer  }  }"; | If you want to draw a spline in the background, define it here. If Color is ‘grey’ then GreyShade = -1, otherwise Color is converted to a grey scale, value for GreyShade between 0 and 1. Add more lines by adding lijn2 etc. |
| $axis\_definition = "{  AxisArrowAngle: integer,  AxisArrowLineColor: string,  AxisArrowLineThickness: integer,  AxisArrowSize: integer,  AxisLineColor: string,  AxisLineThickness: integer,  x\_axis\_position: 'auto',  xAxisArrow: boolean,  xAxisFlipped: boolean,  xAxisName: string,  xAxisNameFontColor: string,  xAxisNameFontSize: integer,  xAxisNameHorizontal: integer,  xAxisNameJustification: string,  xAxisNameVertical: integer,  xLabelColor: string,  xLabelFontSize: integer,  xLabelJustification: string,  xLabelNumberPrecision: float,  xLabelPositionHorizontal: integer,  xLabelPositionVertical: integer,  xLabelShowZero: boolean,  y\_axis\_position: string,  yAxisArrow: boolean,  yAxisFlipped: boolean,  yAxisName: string,  yAxisNameFontColor : string,  yAxisNameFontSize : integer,  yAxisNameHorizontal: integer,  yAxisNameJustification: string,  yAxisNameOrientation: integer,  yAxisNameVertical: integer,  yLabelColor: string,  yLabelFontSize: integer,  yLabelJustification: string,  yLabelNumberPrecision: float,  yLabelPositionHorizontal: integer,  yLabelPositionVertical: integer,  yLabelShowZero: boolean  }"; | Here all the variables for the axes are defined:   * Axis corresponds to both X- and Y-axis   + AxisArrow is the arrow shape at the end of the Axis   + AxisLine is the axis itself * xAxis corresponds to only the X-axis * yAxis corresponds to only the Y-axis * xLabel corresponds to the numeric values of the X-axis label (major grid lines) * yLabel corresponds to the numeric values of the Y-axis label (major grid lines) * x- and y\_axis\_position determine where the axis should be drawn (left, right, top or bottom) |
| $major\_grid\_lines = "{  xStep: float,  yStep: float,  lineWidth: float,  lineColor: float,  checkmark\_offset: integer,  checkmark\_color: string,  checkmark\_width: float  }"; | xStep and yStep define the axis step size for the grid line. For example, xStep = 1 draws the vertical gridlines at each position along the X-axis where the x-value is a multiple of 1, starting from axes variable *xmin* (i.e., 0, 1 ,2… or 0.5, 1.5, 2.5… if X-axis starts with 0.5). Generally, xStep and yStep for minor grid lines are smaller than for major grid lines and an integer multiple of the major xStep or yStep (i.e., if xStep major = 1 then xStep minor is e.g., 0.1/0.25/0.5. |
| $minor\_grid\_lines = "{  xStep: float,  yStep: float,  lineWidth: float,  lineColor: float,  checkmark\_offset: integer,  checkmark\_color: string,  checkmark\_width: float  }"; |
| $interaction\_settings = "{  deltax: integer,  deltay: integer,  draw\_step: integer,  spline\_color: string,  spline\_width: float  }"; | Define the settings for interaction with the canvas:   * deltax and deltay define the distance of the added points when pressing Min/Max button * draw\_step defines the pixel step when adding points after interpolation for the spline |

Appendix B – Draw\_axis() helper functions

**draw\_line()**

*/\*\**

*\* Draw a line. Used to draw the major and minor gridlines,*

*\* and major and minor axis checkmarks.*

*\* @param {Integer} x1*

*\* @param {Integer} y1*

*\* @param {Integer} x2*

*\* @param {Integer} y2*

*\* @param {Integer} width*

*\* @param {String} colour*

*\*/*

function draw\_line(*x1*, *y1*, *x2*, *y2*, *width*, *colour*) {

var line = new scope.Path([new scope.Point(*x1*, *y1*), new scope.Point(*x2*, *y2*)]);

    line.strokeWidth = *width*;

    line.strokeColor = new scope.Color(*colour*) ;

    PermanentElements.addChild(line);

};

**draw\_arrow()**

*/\*\**

*\* Draw a line with an arrow top. Used for the axes.*

*\* @param {PaperScope.Point} startPoint - Paper.js Point object*

*\* @param {PaperScope.Point} endPoint - Paper.js Point object*

*\*/*

function draw\_arrow(*startPoint*, *endPoint*) {

*/\* add axis line \*/*

    var axisLine = new scope.Path();

    axisLine.strokeColor = axis\_definition.AxisLineColor;

    axisLine.strokeWidth = axis\_definition.AxisLineThickness ;

    axisLine.add(*startPoint*);

    axisLine.add(*endPoint*);

    PermanentElements.addChild(axisLine);

*/\* add arrow head \*/*

    var vector = *endPoint*.subtract(*startPoint*);

    vector.length = axis\_definition.AxisArrowSize;

    var vectorItem = new scope.Path([

*endPoint*.add(vector.rotate(axis\_definition.AxisArrowAngle)),

*endPoint*,

*endPoint*.add(vector.rotate(-axis\_definition.AxisArrowAngle))

    ]);

    vectorItem.strokeWidth = axis\_definition.AxisArrowLineThickness;

    vectorItem.strokeColor = axis\_definition.AxisArrowLineColor;

    PermanentElements.addChild(vectorItem);

};

**draw\_label\_text()**

*/\*\**

*\* Draw the text for the axis labels (not axis name).*

*\* @param {Integer} digit - integer from the iteration*

*\* @param {Integer} x1 - X-coordinate to draw*

*\* @param {Integer} y1 - Y-coordinate to draw*

*\* @param {String} Justification - Text justification (left, right, center)*

*\* @param {String} Color - Text color*

*\* @param {Integer} FontSize - Text font size*

*\* @param {Boolean} ShowZero - Show zero value at origin*

*\* @param {Number} NumberPrecision - Decimal precision (0 for integers)*

*\*/*

function draw\_label\_text(*digit*, *x1*, *y1*, *Justification*, *Color*, *FontSize*, *ShowZero*, *NumberPrecision*) {

var text = new scope.PointText(new scope.Point(*x1*, *y1*));

    text.justification = *Justification*;

    text.fillColor = *Color*;

    text.fontSize = *FontSize*;

    if (!*ShowZero*) {

    text.content = *digit*.toFixed(*NumberPrecision*);

    }

    PermanentElements.addChild(text);

};

**even\_fix()**

*/\*\**

*\* In Chrome and Edge gridlines are blurred as they are drawn over 2 pixels.*

*\* Fix by shifting gridlines 0.5 pixel.*

*\* !CAUTION! Fix is unsafe because it checks userAgent string, which is not "unique".*

*\* @param {Number} number - current pixels*

*\* @param {Number} value - pixel shift*

*\* @returns*

*\*/*

function even\_fix(*number*, *value*) {

var isChromeEdge = false;

    var agent = navigator.userAgent;

    if ((agent.indexOf("Chrome") !== -1) || (agent.indexOf("Edg") !== -1)) {

    console.log("Chrome/Edge browser identified to fix draw line pixel bug.")

    isChromeEdge = true;

    }

    if (isChromeEdge) {

    return *number* + *value*;

    }

    else {

    return *number*;

    }

};

Appendix C – Determine axes browser coordinates

**X-axis position**

if (axis\_definition.y\_axis\_position != "auto" ) {

if (axis\_definition.y\_axis\_position == "left") {

    y\_axis\_x\_coordinate = canvasDef.hPad;

    }

    else if (axis\_definition.y\_axis\_position == "right") {

    y\_axis\_x\_coordinate = canvasDef.width - canvasDef.hPad;

    }

    else {

    console.log("y\_axis\_position undefined, selector: " + axis\_definition.y\_axis\_position + " unknown. [auto, left, right]");

    }

}

*/\* determine location when option is 'auto' \*/*

else {

*/\* x\_axis start is >=0 AND axis end is >0 AND x\_axis is not flipped \*/*

   if (axes[0] >= 0 && axes[1] > 0 && !axis\_definition.xAxisFlipped) {

    y\_axis\_x\_coordinate = canvasDef.hPad;

  }

*/\* x\_axis start is >=0 AND axis end is >0 AND x\_axis is flipped \*/*

   else if (axes[0] >= 0 && axes[1] > 0 && axis\_definition.xAxisFlipped) {

    y\_axis\_x\_coordinate = canvasDef.width - canvasDef.hPad;

  }

*/\* x\_axis start is <0 AND axis end is <=0 AND axis is not flipped \*/*

  else if (axes[0] < 0 && axes[1] <= 0 && !axis\_definition.xAxisFlipped) {

    y\_axis\_x\_coordinate = canvasDef.width - canvasDef.hPad;

   }

*/\* x\_axis start is <0 AND axis end is <=0 AND axis is flipped \*/*

   else if (axes[0] < 0 && axes[1] <= 0 && axis\_definition.xAxisFlipped) {

    y\_axis\_x\_coordinate = canvasDef.hPad;

   }

*/\* x\_axis crosses 0 and is not flipped, set to x = 0 coordinate \*/*

   else if (!axis\_definition.xAxisFlipped) {

       y\_axis\_x\_coordinate =  Math.abs(axes[0]) / major\_grid\_lines.xStep \* majorX\_BrowserStep + canvasDef.hPad;

   }

*/\* x\_axis crosses 0 and is flipped, set to x = 0 coordinate \*/*

   else {

      y\_axis\_x\_coordinate = -Math.abs(axes[0]) / major\_grid\_lines.xStep \* majorX\_BrowserStep - canvasDef.hPad + canvasDef.width;

   }

}

**Y-axis position**

if (axis\_definition.x\_axis\_position != "auto" ) {

if (axis\_definition.x\_axis\_position == "top") {

    x\_axis\_y\_coordinate = canvasDef.vPad;

    }

    else if (axis\_definition.x\_axis\_position == "bottom") {

    x\_axis\_y\_coordinate = canvasDef.height - canvasDef.vPad;

    }

    else {

    console.log("x\_axis\_position undefined, selector: " + axis\_definition.x\_axis\_position + " unknown. [auto, top, bottom]");

    }

}

*/\* determine location when option is 'auto' \*/*

else {

*/\* y\_axis start is >=0 AND end is >0 AND axis is not flipped \*/*

    if (axes[2] >= 0 && axes[3] > 0 && !axis\_definition.yAxisFlipped) {

    x\_axis\_y\_coordinate = canvasDef.height - canvasDef.vPad;

    }

*/\* y\_axis start is >=0 AND end is >0 AND axis is flipped \*/*

    else if (axes[2] >= 0 && axes[3] > 0 && axis\_definition.yAxisFlipped) {

    x\_axis\_y\_coordinate = canvasDef.vPad;

    }

*/\* y\_axis start is <0 AND end is <=0 AND axis is not flipped \*/*

    else if (axes[2] < 0 && axes[3] <= 0 && !axis\_definition.yAxisFlipped) {

    x\_axis\_y\_coordinate = canvasDef.vPad;

    }

*/\* y\_axis start is <0 AND end is <=0 AND axis is flipped \*/*

    else if (axes[2] < 0 && axes[3] <= 0 && axis\_definition.yAxisFlipped) {

    x\_axis\_y\_coordinate = canvasDef.height - canvasDef.vPad;

    }

*/\* y\_axis crosses zero and is not flipped, set to y = 0 coordinate \*/*

    else if (!axis\_definition.yAxisFlipped) {

    x\_axis\_y\_coordinate =  Math.abs(axes[3])/major\_grid\_lines.yStep \* majorY\_BrowserStep + canvasDef.vPad;

    }

*/\* y\_axis crosses zero and is flipped, set to y = 0 coordinate \*/*

    else {

    x\_axis\_y\_coordinate = -Math.abs(axes[3])/major\_grid\_lines.yStep \* majorY\_BrowserStep - canvasDef.vPad + canvasDef.height;

    }

}

Appendix D – Drawing the background graph

**X-axis, major vertical gridlines and check marks & X-axis labels**

if (!axis\_definition.xAxisFlipped) {

*/\* Draw x\_axis, not flipped. Extend line 0.5\*hPad beyond graph area \*/*

    if (axis\_definition.xAxisArrow) {

        draw\_arrow(new scope.Point(canvasDef.hPad, x\_axis\_y\_coordinate), new scope.Point(canvasDef.width - canvasDef.hPad/2, x\_axis\_y\_coordinate));

    }

    var x\_temp = even\_fix(canvasDef.hPad, 0.5);

}

else {

*/\* Draw x\_axis, flipped. Extend line 0.5\*hPad beyond graph area \*/*

    if (axis\_definition.xAxisArrow) {

        draw\_arrow(new scope.Point(canvasDef.width - canvasDef.hPad, x\_axis\_y\_coordinate), new scope.Point(canvasDef.hPad/2, x\_axis\_y\_coordinate));

    }

    var x\_temp = even\_fix(canvasDef.width - canvasDef.hPad, -0.5);

}

for (var i = axes[0]; i <= axes[1] ; i = i + major\_grid\_lines.xStep) {

*/\* Draw major vertical grid line \*/*

    draw\_line(x\_temp, canvasDef.vPad, x\_temp, canvasDef.height - canvasDef.vPad, major\_grid\_lines.lineWidth, major\_grid\_lines.lineColor);

*/\* Draw major vertical checkmark \*/*

    draw\_line(x\_temp, x\_axis\_y\_coordinate + major\_grid\_lines.checkmark\_offset, x\_temp, x\_axis\_y\_coordinate - major\_grid\_lines.checkmark\_offset,

        major\_grid\_lines.checkmark\_width, major\_grid\_lines.checkmark\_color);

*/\* Draw x\_axis labels \*/*

    draw\_label\_text(i, x\_temp + axis\_definition.xLabelPositionHorizontal, x\_axis\_y\_coordinate + axis\_definition.xLabelPositionVertical,

        axis\_definition.xLabelJustification, axis\_definition.xLabelColor, axis\_definition.xLabelFontSize, axis\_definition.xLabelShowZero,

        axis\_definition.xLabelNumberPrecision);

    if (!axis\_definition.xAxisFlipped) {

        x\_temp = (x\_temp + majorX\_BrowserStep);

    }

    else {

        x\_temp = (x\_temp - majorX\_BrowserStep);

    }

}

**Y-axis, major horizontal gridlines and check marks & Y-axis labels**

if (!axis\_definition.yAxisFlipped) {

*/\* Draw y\_axis, not flipped. Extend line 0.5\*vPad beyond graph area \*/*

    if (axis\_definition.yAxisArrow) {

        draw\_arrow(new scope.Point(y\_axis\_x\_coordinate, canvasDef.height - canvasDef.vPad), new scope.Point(y\_axis\_x\_coordinate, canvasDef.vPad/2));

    }

    var y\_temp = even\_fix(canvasDef.height - canvasDef.vPad, 0.5);

}

else {

*/\* Draw y\_axis, flipped. Extend line 0.5\*vPad beyond graph area \*/*

    if (axis\_definition.yAxisArrow){

        draw\_arrow(new scope.Point(y\_axis\_x\_coordinate, canvasDef.vPad), new scope.Point(y\_axis\_x\_coordinate, canvasDef.height-canvasDef.vPad/2));

    }

    var y\_temp = even\_fix(canvasDef.vPad, 0.5);

}

for (var i = axes[2]; i <= axes[3] ; i = i + major\_grid\_lines.yStep) {

*/\* Draw major horizontal grid line \*/*

    draw\_line(canvasDef.hPad, y\_temp, canvasDef.width - canvasDef.hPad, y\_temp, major\_grid\_lines.lineWidth, major\_grid\_lines.lineColor);

*/\* Draw major horizontal checkmark \*/*

    draw\_line(y\_axis\_x\_coordinate + major\_grid\_lines.checkmark\_offset, y\_temp, y\_axis\_x\_coordinate - major\_grid\_lines.checkmark\_offset, y\_temp,

        major\_grid\_lines.checkmark\_width, major\_grid\_lines.checkmark\_color);

*/\* Draw y\_axis labels \*/*

    draw\_label\_text(i, y\_axis\_x\_coordinate + axis\_definition.yLabelPositionHorizontal, y\_temp + axis\_definition.yLabelPositionVertical,

        axis\_definition.yLabelJustification, axis\_definition.yLabelColor, axis\_definition.yLabelFontSize ,axis\_definition.yLabelShowZero,

        axis\_definition.yLabelNumberPrecision);

    if (!axis\_definition.yAxisFlipped) {

        y\_temp = y\_temp - majorY\_BrowserStep;

    }

    else {

        y\_temp = y\_temp + majorY\_BrowserStep;

    }

}

**Minor vertical gridlines and check marks & minor horizontal gridlines and check marks**

x\_temp = even\_fix(canvasDef.hPad, 0.5);

for (var i = axes[0]; i <= axes[1] ; i = i + minor\_grid\_lines.xStep) {

*/\* Draw minor vertical grid line \*/*

    draw\_line(x\_temp, canvasDef.vPad, x\_temp, canvasDef.height - canvasDef.vPad, minor\_grid\_lines.lineWidth, minor\_grid\_lines.lineColor);

*/\* Draw minor vertical checkmark (along x-axis) \*/*

    draw\_line(x\_temp, x\_axis\_y\_coordinate + minor\_grid\_lines.checkmark\_offset, x\_temp, x\_axis\_y\_coordinate - minor\_grid\_lines.checkmark\_offset,

        minor\_grid\_lines.checkmark\_width, minor\_grid\_lines.checkmark\_color);

    x\_temp = x\_temp + minorX\_BrowserStep;

}

y\_temp = even\_fix(canvasDef.height - canvasDef.vPad, 0.5);

for (var i = axes[2]; i <= axes[3] ; i = i + minor\_grid\_lines.yStep) {

*/\* Draw minor horizontal grid line \*/*

    draw\_line(canvasDef.hPad, y\_temp, canvasDef.width - canvasDef.hPad, y\_temp, minor\_grid\_lines.lineWidth, minor\_grid\_lines.lineColor);

*/\* Draw minor horizontal checkmark (along y-axis) \*/*

    draw\_line(y\_axis\_x\_coordinate + minor\_grid\_lines.checkmark\_offset, y\_temp, y\_axis\_x\_coordinate - minor\_grid\_lines.checkmark\_offset, y\_temp,

        minor\_grid\_lines.checkmark\_width, minor\_grid\_lines.checkmark\_color);

    y\_temp = y\_temp - minorY\_BrowserStep;

}

**X- and Y-axis names**

if (axis\_definition.xAxisName != "") {

    var text = new scope.PointText(new scope.Point(canvasDef.width/2 + axis\_definition.xAxisNameHorizontal, x\_axis\_y\_coordinate+axis\_definition.xAxisNameVertical));

    text.justification = axis\_definition.xAxisNameJustification;

    text.fillColor = axis\_definition.xAxisNameFontColor;

    text.fontSize = axis\_definition.xAxisNameFontSize;

    text.content = axis\_definition.xAxisName;

    PermanentElements.addChild(text);

}

if (axis\_definition.yAxisName != "") {

    var text = new scope.PointText(new scope.Point(y\_axis\_x\_coordinate + axis\_definition.yAxisNameHorizontal, canvasDef.height/2 + axis\_definition.yAxisNameVertical));

    text.rotate(axis\_definition.yAxisNameOrientation);

    text.justification = axis\_definition.yAxisNameJustification;

    text.fillColor = axis\_definition.yAxisNameFontColor;

    text.fontSize = axis\_definition.yAxisNameFontSize;

    text.content =  axis\_definition.yAxisName;

    PermanentElements.addChild(text);

}

Appendix E – Drawing the background line

for (var property in backgroundlines) {

    if (backgroundlines.hasOwnProperty(property)) {

        var lineObject = backgroundlines[property];

        var x\_val = lineObject.x;

        var y\_val = lineObject.y;

        var lineSpline = new MonotonicCubicSpline(x\_val, y\_val);

        var linePath = new scope.Path();

        linePath.strokeWidth = backgroundlines.lineThickness;

*/\* If GreyShade < 0, use lineColor for strokeColor.*

*Otherwise, ignore lineColor and use GreyShade \*/*

        if (lineObject.lineColorGreyShade < 0) {

            linePath.strokeColor = lineObject.lineColor;

        }

        else {

            linePath.strokeColor = new scope.Color(lineObject.lineColorGreyShade);

        }

*/\* Add points to create background spine \*/*

        if (AxisXtoBrowserX(x\_val[0]) > AxisXtoBrowserX(x\_val[x\_val.length-1])) {

            for (var pointX = AxisXtoBrowserX(x\_val[0]); pointX >= AxisXtoBrowserX(x\_val[x\_val.length-1]); pointX = pointX - interaction\_settings.draw\_step) {

                var temp\_y = lineSpline.interpolate(BrowserXtoAxisX(pointX));

                var temp = new scope.Point(pointX, AxisYtoBrowserY(temp\_y));

                if ((!(lineObject.hasOwnProperty("x\_limit\_max")) || ((lineObject.hasOwnProperty("x\_limit\_max")) && AxisXtoBrowserX(lineObject.x\_limit\_max) <= pointX)) && (!(lineObject.hasOwnProperty("x\_limit\_min")) || ((lineObject.hasOwnProperty("x\_limit\_min")) && AxisXtoBrowserX(lineObject.x\_limit\_min) >= pointX))) {

                        linePath.add(temp);

                }

            }

        }

        else {

            for (var pointX = AxisXtoBrowserX(x\_val[0]); pointX <= AxisXtoBrowserX(x\_val[x\_val.length-1]); pointX = pointX + interaction\_settings.draw\_step) {

                var temp\_y = lineSpline.interpolate(BrowserXtoAxisX(pointX));

                var temp = new scope.Point(pointX, AxisYtoBrowserY(temp\_y));

                if ((!(lineObject.hasOwnProperty("x\_limit\_max")) || ((lineObject.hasOwnProperty("x\_limit\_max")) && AxisXtoBrowserX(lineObject.x\_limit\_max) >= pointX)) &&  (!(lineObject.hasOwnProperty("x\_limit\_min")) || ((lineObject.hasOwnProperty("x\_limit\_min")) && AxisXtoBrowserX(lineObject.x\_limit\_min) <= pointX))) {

                    linePath.add(temp);

                }

            }

        }

        if (!(lineObject.hasOwnProperty("x\_limit\_max")) || ((lineObject.hasOwnProperty("x\_limit\_max")) &&  (lineObject.x\_limit\_max >= x\_val[x\_val.length-1]))) {

            linePath.add(new scope.Point(AxisXtoBrowserX(x\_val[x\_val.length-1]), AxisYtoBrowserY(y\_val[x\_val.length-1])));

        }

        PermanentElements.addChild(linePath);

    }

}

Appendix F – Mouse interaction

function interact() {

tool.onMouseDown = function(*click*) {

.

.

.

};

tool.onMouseDown = function(*click*) {

.

.

.

};

};

**tool.onMouseDown**

tool.onMouseDown = function(*click*) {

    var hitPoint = new scope.Point(*click*.event.offsetX, *click*.event.offsetY);

    var hitResult = DrawnPoints.hitTest(hitPoint, hitOptions);

    if (!hitResult) {

    if (selected\_x !== null || selected\_y !== null ) {

        selected\_x = null;

            selected\_y = null;

        }

        else {

*/\* First point \*/*

            if (PointsLocation.length == 0) {

            PointsLocation.push(hitPoint);

                FittedSplineBrowserY.push(*click*.event.offsetY);

                FittedSplineBrowserX.push(*click*.event.offsetX);

            }

*/\* There are points \*/*

            else {

            for (var i = 0 ; i < PointsLocation.length ; i++) {

*/\* Insert point in array \*/*

                    if (*click*.event.offsetX < PointsLocation[i].x) {

                    PointsLocation.splice(i, 0, hitPoint);

                        FittedSplineBrowserX.splice(i, 0, *click*.event.offsetX);

                        FittedSplineBrowserY.splice(i, 0, *click*.event.offsetY);

                        break;

                    }

*/\* New point at the end, append point \*/*

                    else if (i == PointsLocation.length - 1) {

                    PointsLocation.splice(PointsLocation.length, 0, hitPoint);

                        FittedSplineBrowserX.splice(PointsLocation.length, 0, *click*.event.offsetX);

                        FittedSplineBrowserY.splice(PointsLocation.length, 0, *click*.event.offsetY);

                        break;

                    }

                }

            }

        }

    }

    else {

    selected\_x = *click*.event.offsetX;

        selected\_y = *click*.event.offsetY;

    }

    draw\_spline();

};

**tool.onMouseDrag**

tool.onMouseDrag = function(*click*) {

mouseStartLocation = *click*.point;

    var mouseMovedDistance = *click*.delta;

    var results = [];

    for (var i = 0 ; i < PointsLocation.length ; i++) {

*/\* Check if the dragged point is close to an already existing point,*

*push the points to "results" \*/*

        if (mouseStartLocation.getDistance(PointsLocation[i]) < hitOptions.tolerance) {

            results.push(i);

            console.log(results);

        }

    }

*/\* Remove existing point by splicing \*/*

    for (i = (results.length-1) ; i >= 1 ; i--) {

        PointsLocation.splice(results[i], 1);

        FittedSplineBrowserX.splice(results[i], 1);

        FittedSplineBrowserY.splice(results[i], 1);

    }

PointsLocation[results[0]] = new scope.Point(mouseStartLocation.x + mouseMovedDistance.x,mouseStartLocation.y + mouseMovedDistance.y);

    FittedSplineBrowserY[results[0]] = mouseStartLocation.y + mouseMovedDistance.y;

    FittedSplineBrowserX[results[0]] = mouseStartLocation.x + mouseMovedDistance.x;

    draw\_spline();

};

Appendix G – Button functionality

function buttons(){

var delPoint = $('#delPoint');

var delAll = $('#delAll');

var buttonMin = $('#buttonMin');

var buttonMax = $('#buttonMax');

var toggleContrast = $('#toggleContrast');

var gridMajor = $('#gridMajor');

var gridMinor = $('#gridMinor');

.

.

.

};

**Delete button**

delPoint.click(function() {

*/\* loops through all the drawn points \*/*

    for (points = 0 ; points < PointsLocation.length ; points++) {

*/\* if difference between (x,y) coordinate of point and (x,y)*

*coordinate of selected (clicked on screen) is less than 10,*

*point is found and remove that point from array \*/*

        if ((Math.abs(PointsLocation[points].x - selected\_x) < 10) && (Math.abs(PointsLocation[points].y - selected\_y) < 10)) {

        PointsLocation.splice(points, 1);

            pathsPointsfitsX.splice(points, 1);

            pathsPointsfitsY.splice(points, 1);

        }

    }

*/\* deselect point and draw new spline \*/*

    selected\_x = null;

    selected\_y = null ;

    draw\_spline();

});

**Delete All button**

delAll.click(function() {

*/\* splice(0, .length) removes all items from array \*/*

    PointsLocation.splice(0, PointsLocation.length);

    pathsPointsfitsX.splice(0, pathsPointsfitsX.length);

    pathsPointsfitsY.splice(0, pathsPointsfitsY.length);

    drawnPoints.removeChildren();

    splinePoints.removeChildren();

*/\* draw spline \*/*

    draw\_spline();

});

**Min button**

buttonMin.click(function() {

*/\* loops through all the drawn points \*/*

    for (i = 0 ; i < PointsLocation.length ; i++ ) {

*/\* if difference between (x,y) coordinate of point and (x,y) coordinate where clicked on screen*

*is less than hitOptions.tolerance, point is found and remove that point from array \*/*

if ((Math.abs(PointsLocation[i].x - selected\_x) < hitOptions.tolerance) && (Math.abs(PointsLocation[i].y - selected\_y) < hitOptions.tolerance)) {

*/\* insert point before selected point, move point w.r.t. selected point by deltax and deltay \*/*

            PointsLocation.splice(i,0, new scope.Point(PointsLocation[i].x - interaction\_settings.deltax, PointsLocation[i].y - interaction\_settings.deltay ));

            FittedSplineBrowserX.splice(i, 0, PointsLocation[i].x - interaction\_settings.deltax );

            FittedSplineBrowserY.splice(i, 0, PointsLocation[i].y - interaction\_settings.deltay);

*/\* insert point after selected point, move point w.r.t. selected point by deltax and deltay \*/*

            PointsLocation.splice(i+2, 0, new scope.Point(PointsLocation[i].x + interaction\_settings.deltax, PointsLocation[i].y - interaction\_settings.deltay ) );

            FittedSplineBrowserX.splice(i+2, 0, PointsLocation[i].x + interaction\_settings.deltax);

            FittedSplineBrowserY.splice(i+2, 0, PointsLocation[i].y - interaction\_settings.deltay);

            break;

        }

    }

*/\* deselect point and draw new spline \*/*

    selected\_x = null;

    selected\_y = null;

    draw\_spline();

});

**Max button**

buttonMax.click(function() {

*/\* loops through all the drawn points \*/*

    for (i = 0 ; i < PointsLocation.length ; i++ ){

*/\* if difference between (x,y) coordinate of point and (x,y) coordinate where clicked on screen*

*is less than hitOptions.tolerance, point is found and remove that point from array \*/*

        if ((Math.abs(PointsLocation[i].x - selected\_x) < hitOptions.tolerance) && (Math.abs(PointsLocation[i].y - selected\_y) < hitOptions.tolerance)) {

*/\* insert point before selected point, move point w.r.t. selected point by deltax and deltay \*/*

            PointsLocation.splice(i,0, new scope.Point(PointsLocation[i].x - interaction\_settings.deltax, PointsLocation[i].y + interaction\_settings.deltay ));

            FittedSplineBrowserX.splice(i, 0, PointsLocation[i].x - interaction\_settings.deltax);

            FittedSplineBrowserY.splice(i, 0, PointsLocation[i].y + interaction\_settings.deltay);

*/\* insert point after selected point, move point w.r.t. selected point by deltax and deltay \*/*

            PointsLocation.splice(i+2, 0, new scope.Point(PointsLocation[i].x + interaction\_settings.deltax, PointsLocation[i].y + interaction\_settings.deltay ));

            FittedSplineBrowserX.splice(i+2, 0, PointsLocation[i].x + interaction\_settings.deltax);

            FittedSplineBrowserY.splice(i+2, 0, PointsLocation[i].y + interaction\_settings.deltay);

            break;

        }

    }

*/\* deselect point and draw new spline \*/*

    selected\_x = null;

    selected\_y = null;

    draw\_spline();

});

**Toggle contrast settings button**

toggleContrast.click(function() {

*/\* get div element with id: contrast \*/*

    var div\_contrast = document.getElementById('contrast');

if (div\_contrast.style.display === 'none') {

    div\_contrast.style.display = 'block';

    }

    else {

    div\_contrast.style.display = 'none';

    }

});

**Major gridline slider**

gridMajor.click(function() {

    PermanentElements.removeChildren();

    major\_grid\_lines.lineColor = gridMajor.val()/10;

    draw\_axis();

});

**Minor gridline slider**

gridMinor.click(function() {

    PermanentElements.removeChildren();

    minor\_grid\_lines.lineColor = gridMinor.val()/10;

    draw\_axis();

});

1. A *resize* attribute is added to the <canvas> element. While this is not a HTML5 supported attribute, for yet unknown reasons this does ensure the SketchApp fits the canvas. [↑](#footnote-ref-1)