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.

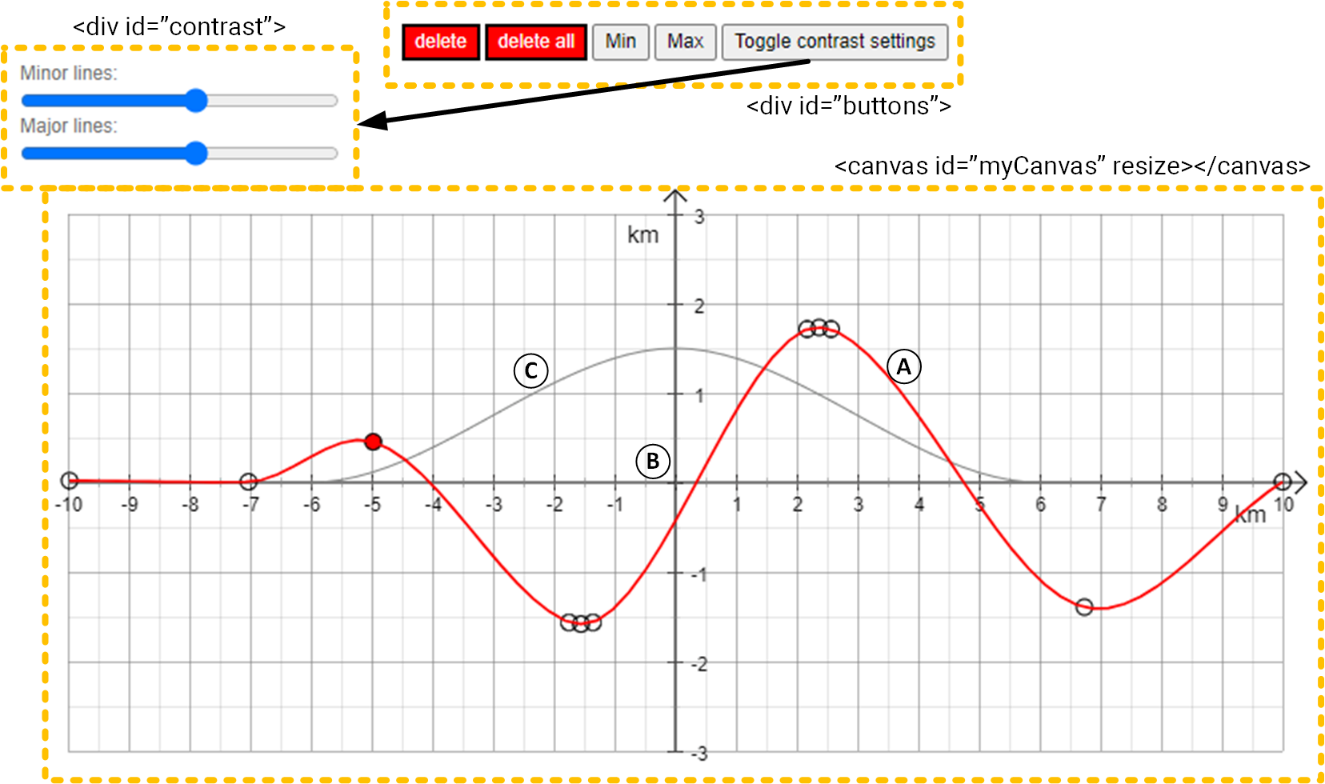


Figure 1: SketchApp example. (A) Spline (B) Background consisting of axes with labels and a grid (C) Spline as background

## 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 1). 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. |

## Background

When the SketchApp is build, an empty graph is created first, from now referred to as background. The background consists of:

* x-axis & y-axis lines
* x-axis label & y-axis labels
* x-axis major & minor tickmarks
* x-axis major & minor tickmark labels
* major & minor gridlines
* a spline (optional)

A Paper.js Group() object called *PermanentElements* is defined. Lines and text are defined using the Paper.js Path() and PointText() objects, respectively. All the objects are added as child objects to *PermanentElements*. When done, *PermanentElements* is added to the *activeLayer* of the scope, drawing the objects to the canvas.

Optionally, a spline can be added to *PermanentElements* using a Path() object. How this is done is explained in section “Drawing the spline”.

## Interaction

Interaction is possible in two ways: (1) *onMouseDown* and (2) *onMouseDrag*. When the user clicks (*onMouseDown*) on the canvas, a Paper.js Point() object is created. The object is then pushed to a *PointsLocation* array, which keeps track of all created points by the user. If a point is created between two already existing points, the point is inserted in the array instead.

If the mouse is dragged when clicked, the position is tracked continuously until released. However, the position of the point in *PointsLocation* is determined by the click event and does not change when dragging. This makes it possible to have a point of which the x-coordinate is larger/smaller than (one of) its neighbour(s). The code for mouse interaction can be found in Appendix B.

## Drawing the spline

So far only the background has been drawn (*PermanentElements*) and the user points tracked (*PointsLocation*). Actual drawing of the spline is done separately and is entirely dependent on the *PointsLocation* array. First two groups are created, *drawnPoints* and *splinePoints*. For each element in *PointsLocation*, Shape.Circle() object is added to *drawnPoints*. Then, a Path(), *lineSpline*, and MonotonicCubicSpline(), *mySpline*, object are created. Interpolation between two elements of *PointsLocation* is done with *mySpline* to build *lineSpline*. Finally, *drawnPoints* and *lineSpline* are added to the *activeLayer*, drawing the points and spline to the canvas.

## Button interaction

The <input> elements with *type=”button”* create simple push buttons to interact with (*click* event handler). The SketchApp uses five buttons:

* Delete Delete the selected point
* Delete all Delete all points
* Min Force the selected point to be a local minimum
* Max Force the selected point to be a local maximum
* Toggle contrast settings Toggle sliders to adjust major and minor gridline opacity

Buttons “Delete” and “Delete all” modify the *PointsLocation* array by removing a single Point() object (“Delete”) or by removing all Point() objects (“Delete all”). The “Min”/”Max” buttons add Point() objects to the *PointsLocation* array, one before and one after the selected element in the array. The added points have a x- and y-offset such that the selected point becomes a local minimum or maximum. The “Toggle contrast settings” shows or hides a nested <div> element with two sliders. The sliders set the *lineColor* attribute of the associated Path() object. The code of all the buttons can be found in Appendix C.

# Designing the background graph

Designing a the background graph is done by adjusting variables in the Algorithm section of a Möbius question. The variables are given and described in Table 1.

Table 4: Möbius question Algorithm variables

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

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 – Mouse interaction

**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();

                };

**onMouseDrag**

tool.onMouseDrag = function(*click*) {

                    mouseStartLocation = *click*.point;

                    var mouseMovedDistance = *click*.delta;

                    var results =[];

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

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

*/\* check if it's only one point. If more than 1 point, remove them \*/*

                            results.push(i);

                        }

                    }

                    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 C – Button functionality

**Delete button**

var delPoint = $('#delPoint');

*/\* delete point \*/*

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**

var delAll = $('#delAll');

*/\* delete all points \*/*

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**

var buttonMin = $('#buttonMin');

*/\* min function \*/*

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

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

*/\* 'temporarily' save y-coordinate of selected point \*/*

                            var tempy = PointsLocation[i].y;

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

                            PointsLocation.splice(i,0, new scope.Point(selected\_x - interaction\_settings.deltax, tempy - interaction\_settings.deltay ));

                            FittedSplineBrowserX.splice(i, 0, selected\_x - interaction\_settings.deltax );

                            FittedSplineBrowserY.splice(i, 0, tempy - 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(selected\_x + interaction\_settings.deltax, tempy - interaction\_settings.deltay ) );

                            FittedSplineBrowserX.splice(i+2, 0, selected\_x + interaction\_settings.deltax);

                            FittedSplineBrowserY.splice(i+2, 0, tempy - interaction\_settings.deltay);

                            break;

                        }

                    }

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

                    selected\_x = null;

                    selected\_y = null ;

                    draw\_spline();

                });

**Max button**

var buttonMax = $('#buttonMax');

*/\* max function \*/*

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

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

*/\* 'temporarily' save y-coordinate of selected point \*/*

                            var tempy = PointsLocation[i].y;

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

                            PointsLocation.splice(i,0, new scope.Point(selected\_x - interaction\_settings.deltax, tempy + interaction\_settings.deltay ));

                            FittedSplineBrowserX.splice(i, 0, selected\_x - interaction\_settings.deltax);

                            FittedSplineBrowserY.splice(i, 0, tempy + 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(selected\_x + interaction\_settings.deltax, tempy + interaction\_settings.deltay ));

                            FittedSplineBrowserX.splice(i+2, 0, selected\_x + interaction\_settings.deltax);

                            FittedSplineBrowserY.splice(i+2, 0, tempy + interaction\_settings.deltay);

                            break;

                        }

                    }

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

                    selected\_x = null;

                    selected\_y = null ;

                    draw\_spline();

                });

**Toggle contrast settings button**

var cont = $('#toggleContrast');

*/\* toggle contrast settings \*/*

cont.click(function() {

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

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

*/\* Default style.display is 'none', thus the div is hidden.*

*On click style.display switches between 'block' and 'none'.*

*The 'block' type ensures the display uses the entire line and not*

*inline. \*/*

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

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

    }

    else {

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

    }

});

var gridMajor = $('#gridMajor');

*/\* major grid lines slider \*/*

gridMajor.click(function() {

*/\* clears #myCanvas \*/*

    scope.project.clear();

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

*/\* redraw axis \*/*

    draw\_axis();

*/\* draw spline \*/*

    draw\_spline();

});

var gridMinor = $('#gridMinor');

*/\* minor grid lines slider \*/*

gridMinor.click(function() {

*/\* clears #myCanvas \*/*

    scope.project.clear();

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

*/\* redraw axis \*/*

    draw\_axis();

*/\* draw spline \*/*

    draw\_spline();

});

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)