

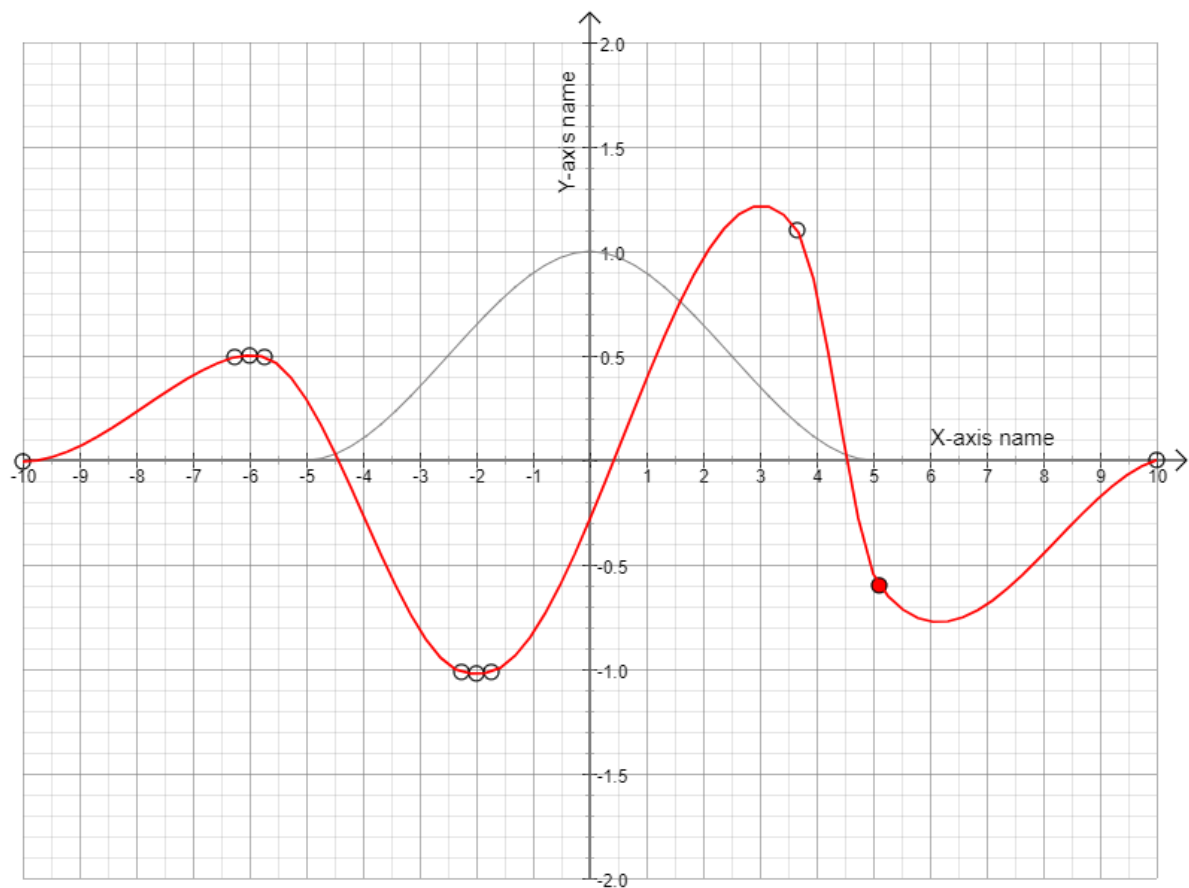
SketchApp

The interactive spline drawing tool for Möbius

DOCUMENTATION

v1.1

20 July 2023



SketchApp

v3.1

20 July 2023

Anatoly Ilin & Mario van den Berg

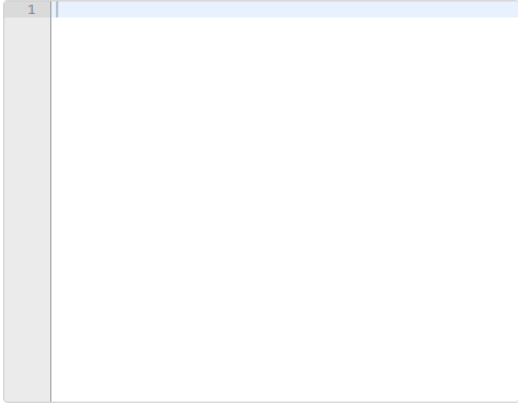
Table of Contents

1	Introduction	2
2	SketchApp overview	3
2.1	Setting up the iframe	4
2.2	Defining Algorithm variables	4
2.3	Setting up the Question JavaScript	5
3	Building the SketchApp	6
3.1	Main code	6
3.1.1	Initialisation	6
3.1.2	Behaviour	8
3.1.3	Conversion functions	8
3.2	Building the background graph	9
3.2.1	Helper functions and canvas resizing	9
3.2.2	Determine axis location on canvas	10
3.2.3	Drawing the axes, gridlines and labels	10
3.2.4	Finishing up	12
3.3	Mouse interaction	13
3.3.1	Clicking the mouse button	13
3.3.2	Dragging the mouse	13
3.4	Button functionality	13
3.4.1	Delete buttons	13
3.4.2	Local extreme buttons	13
3.4.3	Toggle contrast settings	13
3.5	Drawing the spline	14
3.6	Creating the response string	16
	Appendix A – Algorithm field	20
	Appendix B – App behaviour	22
	Appendix C – Conversion functions	23
	Appendix D – Draw_axis() helper functions	24
	Appendix E – Determine axes pixel coordinates	27
	Appendix F – Drawing the background graph	29
	Appendix G – Drawing the background line	33
	Appendix H – Mouse interaction	35
	Appendix I – Button functionality	37

1 Introduction

[Möbius](#) is the online STEM learning platform from [DigitalEd](#). One of its features is the [HTML question type](#) that can be used for as a *response area*. When inserting a HTML response area into a question, a HTML `<iframe>` element is created. An iframe is, simply put, used to embed a HTML page in the current one. Using an iframe ensures that the parent page is not affected. When editing a HTML response area, you are prompted with some options, see Table 1.

Table 1: HTML response area edit options

HTML: Weighting: <input type="text" value="1"/> Answer: <input type="text"/> <small>(referenced when grading as \$ANSWER)</small> Grading Code: <input type="text" value="evalb((\$ANSWER)-(\$RESPONSE)=0);"/>	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. <i>Note: for the Sketchapp use "\$answer".</i>
	Grading Code	Used to evaluate the student response (\$RESPONSE) with the correct answer (\$ANSWER). The code must use valid Maple code (and syntax).
	Question HTML	Here you can define HTML code needed to display what you want (e.g., <code><div></code> , <code><script></code> , <code><input></code> and <code><canvas></code>). Note that everything is contained in the inserted <code><iframe></code> .
	Question CSS	Here you can define CSS code to change how your HTML code looks. Note that this is ONLY applied to the inserted <code><iframe></code> .
	Question JavaScript	Requires three functions: <code>initialize(interactiveMode)</code> , <code>setFeedback(response,answer)</code> and <code>getResponse()</code> . 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;
```

2 SketchApp overview

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

- SketchApp_v3.0.js The full code that creates the SketchApp (a.k.a. SketchApp.js).
- paper-full.js [Paper.js](#) is used to enable interaction with the canvas.
- cubic_spline.js Enables Cubic Monotone (Hermite) Interpolation.

The SketchApp is designed to draw interactive splines on top of a graph with some buttons. An example is given in Figure 1. Within the iframe, buttons are placed and a canvas created. On the canvas a background graph is drawn, with an optional background spline. The user can create points on the canvas. Through the points a Monotonic Cubic (Hermite) Interpolated Spline is drawn. The points can be dragged to modify the spline. Using the buttons, points can be deleted and local minima/maxima can be set. An extra button is added that modifies the gridline contrast using sliders the user can interact with.

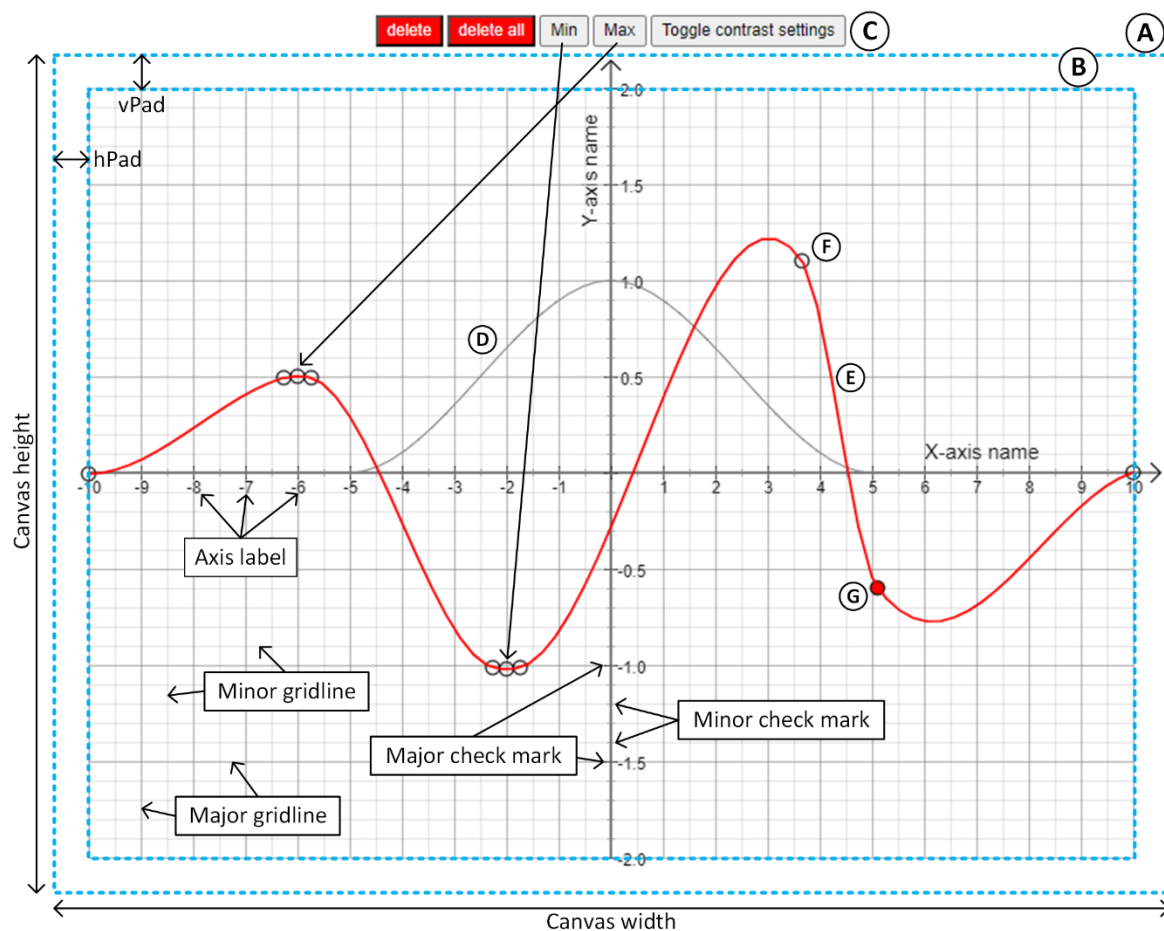


Figure 1: SketchApp example. (A) Canvas (B) Draw area (C) HTML buttons (D) Background spline, optional (E) Monotonic Cubic Interpolated Spline (F) Points created by the user (G) Selected point, to drag or delete.

2.1 Setting up the iframe

The first step is to create a `<div id="buttons">` and `<canvas id="myCanvas">`¹ element in the **Question HTML** part of the response area (Table 1, Figure 2). In the `<div>` element, the necessary HTML buttons are defined 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 `SketchApp.js`, see section 3.4.

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

2.2 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 in Table 2, the full list is provided in [Appendix A](#). Red text is user adjustable.

Table 2: Short list of Algorithm field variables

<code>\$teachermode = "boolean";</code>	If true, the coordinates of the drawn points are displayed in a <code><table></code> element below the SketchApp (but still in the <code><iframe></code>).
<code>\$answer = "[[x1,y1],[x2,y2],...,[xn,yn]]";</code>	The correct answer of the question.
<code>\$axes = "[xmin,xmax,ymin,ymax]";</code>	Set the graph axis limits.
<code>\$answerplot = plotmaple("...");</code>	This displays the correct answer without the need to preview the question.
<code>\$canvasDef = "{...}";</code>	Set canvas width and height and vertical and horizontal padding.
<code>\$backgroundlines = "{...}";</code>	If you want to draw a spline in the background, define it here.
<code>\$X_axis_definition = "{...}";</code>	Defines the axes settings.
<code>\$Y_axis_definition = "{...}";</code>	
<code>\$axes_arrow = "{...}";</code>	
<code>\$major_grid_lines = "{...}";</code>	Defines the gridline settings.
<code>\$minor_grid_lines = "{...}";</code>	
<code>\$interaction_settings = "{...}"</code>	Defines how the spline is displayed and how the Min/Max buttons modify the 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.

2.3 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, but comments and `console.log()` events are omitted. Red text is user specific.

Table 3: Full list of Question Javascript field. Comments and console.log events omitted.

<pre>var teachermode = \$teachermode; var axes = \$axes; var canvasDef = \$canvasDef; var x_axis_definition = \$X_axis_definition; var y_axis_definition = \$Y_axis_definition; var axes_arrow = \$axes_arrow; 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 StringResponse = ""; var gradebook = false;</pre>	<p>Variable teachermode, if true, shows error messages from the code and coordinates of the user clicked points.</p> <p>Variable type defines the behaviour of the app, see further in this table.</p> <p>StringResponse is used for the Grading Code.</p>
<pre>jQuery.getScript('/web/Cie4305005/Public_Html/.../RunApp.js', function(){});</pre>	<p>Define the path to the RunApp.js. Typically, the child class (here: CIE4305005) is used. However, recommended is to use Masterclass but this is not (always) directly accessible.</p>
<pre>function initialize(interactiveMode) { gradebook = !interactiveMode if (gradebook){ jQuery("#buttons").remove(); } };</pre>	<p>When the question is graded, interactiveMode is false and interaction is disabled. Also, the <code><div id="buttons"></code> element is removed from the canvas.</p>
<pre>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"];</pre>	<p>This is a helping variable for SketchApp.js behaviour, see function setFeedback(...) below.</p>
<pre>function setFeedback(response, answer) { if (translations.indexOf(response) >= 0 && answer == null) { if (!gradebook) { type = 1; runApp(response, type); } else { type = 3; runApp(answer, type); } } else if (answer == null) { type = 2; runApp(response, type); } else if (answer != null) { type = 3; runApp(answer, type); } };</pre>	<p>response is either "No answer" (or translation) or student response. answer is either null or correct answer.</p>
<pre>function getResponse() { return StringResponse; };</pre>	<p>SketchApp.js builds StringResponse. Here, StringResponse is returned by the function as \$RESPONSE. In turn \$RESPONSE can be used in the Grading Code.</p>

3 Building the SketchApp

The full code is contained in one function, `runApp(array, type)`. In this chapter code is explained in segments. Each segment builds upon the previous segments, going into evermore detail. Regardless, code is sometimes compressed or minimized for clarity (e.g., comments, `console.log` statements and entire lines of code are omitted). In these cases, the full code can be found in appendices.

3.1 Main code

3.1.1 Initialisation

The required scripts (*paper-full.js* & *cubic_spline_original.js*) are called first. The rest of the code is wrapped inside.

```
function runApp(array, type) {
    jQuery.getScript('path/to/cubic_spline_original.js', function() {
        jQuery.getScript('path/to/paper-full.js', function() {
            ...
        });
    });
}
```

A `PaperScope` and `Tool` object are created to access the `Paper.js` classes. Next, the canvas is given an initial size and an empty project is setup².

```
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 `<table>` element is created within to display the coordinates of 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. Remember they are actually local variables of `runApp()`, but since that is the only function used they act as a sort of global variable for the nested functions.

```
/* Create 3 Paper.js group objects to draw to the screen */
var BackgroundGraph = new scope.Group(); /* stores objects to draw the background */
var UserCircles = new scope.Group(); /* stores Circle objects to draw user points */
var SplineDrawn = new scope.Group(); /* stores the Path object to draw the spline */

/* Create arrays that store information about the user and spline Points */
```

² When using jQuery, a DOM (Document Object Model) element is returned. Passing `[0]` get the actual element.

```

var UserArray = []; /* stores Point objects where the user clicked */
var SplineArray = []; /* stores Point objects from the spline */
var PointsXcoordinate = []; /* stores UserArray Point.x values to create spline */
var PointsYcoordinate = []; /* stores UserArray Point.y values to create spline */

/* Global variables for drawing the background graph */
var y_axis_x_coordinate; /* pixel location of horizontal axis (x-axis) */
var x_axis_y_coordinate; /* pixel location of vertical axis (y-axis) */
var majorX_PixelStep; /* pixel distance between major vertical gridlines */
var majorY_PixelStep; /* pixel distance between major horizontal gridlines */
var minorX_PixelStep; /* pixel distance between minor vertical gridlines */
var minorY_PixelStep; /* pixel distance between minor horizontal gridlines */

/* Global variables for selecting a Point */
var selected_x = null; /* pixel x-coordinate where the user clicked */
var selected_y = null; /* pixel y-coordinate where the user clicked */
var hitPoint; /* Point object where the user clicked */
var hitOptions = /* Options for the hitTest */
{
  fill: false,
  stroke: true,
  segments: true,
  curves: false,
  tolerance: 10
};

/* Boolean for viewing Möbius gradebook */
var viewingGradebook = false;
/* Initialize variable for Monotonic Cubic Interpolation */
var mySplineDraw;
/* String variable to display when teacher mode is true */
var errorMessages = "No errors from the code. If it's not working, check console log.";

```

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 draw to and interact with the canvas, depending on what behaviour (type) the App is assigned. The functions `buttons()`, `interact()`, `draw_spline()` and `create_answer()` define the behaviour:

- `interact()` Defines what mouse interaction is possible, see section 3.3.
- `buttons()` Defines what the HTML buttons do, see section 3.4.
- `draw_spline()` Draws the interpolated spline (and user points), see section 3.5.
- `create_response()` Creates the `$RESPONSE` for Möbius, see section 3.6.

3.1.2 Behaviour

The SketchApp knows three types of behaviour: 1, 2 and 3. The type is set by Möbius, from the function `setFeedback(response, answer)`. If the question is opened for the first time or the student has not yet given a response yet, then `type = 1`. This only occurs when the question is ungraded. As no response has been given, there is no need to draw a spline or create a response string. If the student has given a response, but the question is not yet graded, then `type = 2`. Now also the spline needs to be drawn and the response string created. If the question is ungraded, the response can still be edited. Otherwise, only the spline is drawn. When the question is graded or when the gradebook is viewed, `type = 3` and the correct answer is drawn. The full code and explanation can be found in [Appendix B](#).

```
if (type==1) {
  buttons();
  interact();
}
else if (type==2) {
  ... // here the user response is loaded.
  if (gradebook) {
    draw_spline();
  }
  else {
    buttons();
    interact();
    draw_spline();
    create_response();
  }
}
else if (type==3) {
  ... // here the correct answer is loaded.
  draw_spline();
}
else {
  console.log("Error! ...");
}
```

3.1.3 Conversion functions

The points created by the user are stored as pixel coordinates. For reviewing and grading the responses, it is sometimes preferable to know the corresponding axes coordinates. Vice versa, the correct answer for a question and the optional background spline are given in axes coordinates. To view these on screen, they have to be translated to pixel coordinates. Four functions are created to convert between the different reference frames:

- `PixelXtoAxisX(x_loc)` Convert pixel x-coordinate to axis x-coordinate
- `PixelYtoAxisY(y_loc)` Convert pixel y-coordinate to axis y-coordinate
- `AxisXtoPixelX(x_loc)` Convert axis x-coordinate to pixel x-coordinate
- `AxisYtoPixelY(y_loc)` Convert axis y-coordinate to pixel y-coordinate

The full code for the functions is provided in [Appendix C](#). Note that pixel coordinates are relative to the `<iframe>` element.

3.2 Building the background graph

3.2.1 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 helper functions:

- `Draw_arrow(...)` Draws a line with an arrow tip. Used for the axes.
- `Draw_axis_name(...)` Draw the axis name.
- `Draw_line(...)` Draws a line. Used to draw the major and minor gridlines, and major and minor checkmarks.
- `Draw_label_text(...)` Draws the text for the axes labels.
- `PixelBugFix(...)` A fix for Chrome and Edge displaying gridlines. Possibly extends to more Chromium based browsers.

The code for the above functions, including a more detailed explanation can be found in [Appendix D](#). Before building the background, everything from `BackgroundGraph` is removed first:

```
BackgroundGraph.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 px and hPad = 30 px. The draw area is 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; thus, the graph is incorrect after rounding the numbers.

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. Setting `scope.viewSize` rescales the Paper.js canvas. Lastly, the major gridlines pixel step is calculated.

```
minorX_PixelStep = Math.round((canvasDef.width - canvasDef.hPad*2) / ((Math.abs(axes[1] - axes[0]) / minor_grid_lines.xStep)));
minorY_PixelStep = Math.round((canvasDef.height - canvasDef.vPad*2) / ((Math.abs(axes[3] - axes[2]) / minor_grid_lines.yStep)));

canvasDef.width = minorX_PixelStep * ((Math.abs(axes[1] - axes[0]) / minor_grid_lines.xStep)) + canvasDef.hPad*2;
canvasDef.height = minorY_PixelStep * ((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_PixelStep = Math.round((canvasDef.width - canvasDef.hPad*2) / ((Math.abs(axes[1] - axes[0]) / major_grid_lines.xStep)));
majorY_PixelStep = Math.round((canvasDef.height - canvasDef.vPad*2) / ((Math.abs(axes[3] - axes[2]) / major_grid_lines.yStep)));
```

3.2.2 Determine axis location on canvas

The next step is to determine the pixel coordinates of the X- and Y-axis. As three user options are available, we need to address this in the code. Below part of the code is provided, the full code can be found in [Appendix E](#).

X-axis position

```
if (x_axis_definition.Position != "auto" ) {  
    if (x_axis_definition.Position == "top") {  
        x_axis_y_coordinate = canvasDef.vPad;  
    }  
    else if (x_axis_definition.Position == "bottom") {  
        x_axis_y_coordinate = canvasDef.height - canvasDef.vPad;  
    }  
    else {  
        console.log("Error!...");  
    }  
}  
else {  
    ... // Here the location is determined from the axis domain  
}
```

Y-axis position

```
if (y_axis_definition.Position != "auto" ) {  
    if (y_axis_definition.Position == "left") {  
        y_axis_x_coordinate = canvasDef.hPad;  
    }  
    else if (y_axis_definition.Position == "right") {  
        y_axis_x_coordinate = canvasDef.width - canvasDef.hPad;  
    }  
    else {  
        console.log("Error!...");  
    }  
}  
else {  
    ... // Here the location is determined from the axis domain  
}
```

3.2.3 Drawing the axes, gridlines and labels

Now that the canvas has the correct size and know where the axes need to go the background graph can be drawn. First the axes and their names are drawn. Then, the major vertical gridlines and check marks and the X-axis labels are drawn. Similarly, the major horizontal gridlines and check marks and Y-axis labels are drawn. Lastly, the minor vertical and horizontal gridlines and check marks are drawn. A compressed version of the code is provided below. The full code can be found in [Appendix F](#).

X-axis line and name

```

if (!x_axis_definition.Flipped) {
    if (x_axis_definition.Arrow) {
        draw_arrow(...); // Draws the axis with an arrow head.
    }
    var x_temp = pixelBugFix(canvasDef.hPad, 0.5);
}
else {
    if (x_axis_definition.Arrow) {
        draw_arrow(...); // Draws the axis with an arrow head.
    }
    var x_temp = pixelBugFix(canvasDef.width - canvasDef.hPad, -0.5);
}
var x_axis_text = new scope.PointText(...);
draw_axis_name(x_axis_definition, x_axis_text); // Draws the axis name.

```

Y-axis line and name

```

if (!y_axis_definition.Flipped) {
    if (y_axis_definition.Arrow) {
        draw_arrow(...); // Draws the axis with an arrow head.
    }
    var y_temp = pixelBugFix(canvasDef.height - canvasDef.vPad, 0.5);
}
else {
    if (y_axis_definition.Arrow){
        draw_arrow(...); // Draws the axis with an arrow head.
    }
    var y_temp = pixelBugFix(canvasDef.vPad, 0.5);
}
var y_axis_text = new scope.PointText(...);
draw_axis_name(y_axis_definition, y_axis_text); // Draws the axis name.

```

Major vertical gridlines & checkmarks and X-axis labels

```

for (var i = axes[0] ; i <= axes[1] ; i = i + major_grid_lines.xStep) {
    /* major vertical grid line */
    draw_line(...);
    /* major vertical checkmark */
    draw_line(...);
    /* x_axis labels */
    draw_label_text(...);
    if (!x_axis_definition.Flipped) {
        x_temp = (x_temp + majorX_PixelStep);
    }
    else {
        x_temp = (x_temp - majorX_PixelStep);
    }
}

```

Major horizontal gridlines & checkmarks and Y-axis labels

```
for (var i = axes[2] ; i <= axes[3] ; i = i + major_grid_lines.yStep) {
  /* major horizontal grid line */
  draw_line(...);
  /* major horizontal checkmark */
  draw_line(...);
  /* y_axis labels */
  draw_label_text(...);
  if (!y_axis_definition.Flipped) {
    y_temp = y_temp - majorY_PixelStep;
  }
  else {
    y_temp = y_temp + majorY_PixelStep;
  }
}
```

Minor vertical gridlines & check marks

```
x_temp = pixelBugFix(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_PixelStep;
}
```

Minor horizontal gridlines & check marks

```
y_temp = pixelBugFix(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_PixelStep;
}
```

3.2.4 Finishing up

Optionally background spline(s) can be drawn and is similar to drawing the user spline (section 3.5). The full code can be found in [Appendix G](#). So far, we have added all the Paper.js objects to the group BackgroundGraph. To truly draw these objects to the canvas we add BackgroundGraph to the active layer of the scope:

```
scope.project.activeLayer.addChild(BackgroundGraph);
```

3.3 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.onMouseDown` 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 H](#).

3.3.1 Clicking the mouse button

When the user clicks on the canvas area, the mouse location is stored in a `Point()` object. The coordinates of the object are tested for a hit with the `UserCircles` group. If true, the user clicked on an already 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, a check is done if any points exist in `UserArray`. If no points exist, the coordinates are pushed to `UserArray`. If points already exist the new point is spliced into the array, which includes adding it to the beginning or end of the array. Lastly, we call the `draw_spline()` and `create_response()` functions to respectively redraw the spline and create the response string.

3.3.2 Dragging the mouse

When the mouse is dragged, the starting mouse location and distance (delta) from the start are stored. Each update a check is done if the mouse is near the dragged point. If true, the point coordinates are set to where the mouse is. Thus, when the mouse is too far from the dragged point when its location is updated the point coordinates are not updated. Consequently, there is a limit to how fast a point can be dragged. Also, if the dragged point gets close to another point, that point is removed from `UserArray`.

3.4 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 I](#). Here, only the concepts are described.

```
var delPoint = $('#delPoint'); /* Delete the selected point */
var delAll = $('#delAll'); /* Delete all points */
var localMin = $('#localMin'); /* Force the selected point to be a local minimum */
var localMax = $('#localMax'); /* 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 */
```

3.4.1 Delete buttons

These buttons modify `UserArray`. The “Delete” button deletes the selected point, redraws the spline and recreates the response string. The “Delete all” button removes all points, but also removes all content from the `UserCircles` and `SplineDrawn` groups.

3.4.2 Local extreme buttons

These buttons modify `UserArray`. The buttons insert two more points, one before and one after the selected point. The coordinates of the new points get an offset from the selected point coordinates, defined by `interaction_settings.deltax` and `interaction_settings.deltay`.

3.4.3 Toggle contrast settings

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 is

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()`.

3.5 Drawing the spline

In section 3.1.1 a `<table id="teacher">` element is created when `teachermode` is true. At the start of `draw_spline()`, the contents of the table are created and passed to the webpage.

```
if (teachermode) {
    var tableText = "<table style='width:50%'>";
    tableText = tableText + "<tr> <th> x: </th> <th> y: </th> </tr>";
    for (var i = 0 ; i < UserArray.length ; i++ ) {
        tableText = tableText + "<tr> <td>" + PixelXtoAxisX(UserArray[i].x) +
            "</td> <td>" + PixelYtoAxisY(UserArray[i].y) +
            "</td> </tr>";
    }
    tableText = tableText + "</table>";
    document.getElementById("teacher").innerHTML = errorMessages + "<br/>" + "Drawn points"
    <br/>" + tableText;
}
```

Each time the spline is drawn, the associated variables are cleared first:

```
SplineArray = [];
PointsXcoordinate = [];
PointsYcoordinate = [];
UserCircles.removeChildren();
SplineDrawn.removeChildren();
```

Then a check is done to see if the length of `UserArray` is nonzero. If true, a second check is done to see if the gradebook is NOT viewed. If true, the points clicked by the user are shown by creating `Shape.Circle()` objects. If the user has clicked on an already drawn circle, the fill colour of that circle is set to red. All `Shape.Circle()` objects are drawn via `UserCircles`.

```
if (UserArray.length != 0) {
    /* Draw circles where the user clicked if not viewing gradebook. */
    if (!viewingGradebook) {
        for (var i = 0 ; i < UserArray.length ; i++ ) {
            var circle = new scope.Shape.Circle(UserArray[i],
interaction_settings.circle_radius);
            circle.strokeColor = 'black';
            /* If circle is selected, change fillColor to red */
            if (selected_x != null && selected_y != null &&
hitPoint.getDistance(UserArray[i]) < hitOptions.tolerance) {
                circle.fillColor = 'red';
            }
            UserCircles.addChild(circle);
        }
    }
}
```

The actual spline that is drawn is a single `Path()` object, stored in `SplinePath`. In order to create the points to add to `SplinePath` the coordinates of the `Point()` objects in `UserArray` need to be separated.

```
if (UserArray.length >= 2) {
  /* Draw the spline from the user input. */
  var SplinePath = new scope.Path();
  SplinePath.strokeColor = interaction_settings.spline_color;
  SplinePath.strokeWidth = interaction_settings.spline_width;

  for (var i = 0 ; i < UserArray.length ; i++ ) {
    PointsXcoordinate.push(UserArray[i].x);
    PointsYcoordinate.push(UserArray[i].y);
  }
}
```

A monotonic cubic spline is created for interpolation, `mySplineDraw`. The spline is used to determine the y-coordinate of the interpolated points. The first point is the first point in `UserArray`. Then, the next point is determined by adding the `draw_step` to the starting x-coordinate. The y-coordinate is calculated from the spline. This process continues until the next point is larger than that of the last point in `UserArray`. The last point of `UserArray` is added, finalizing the path to draw.

```
mySplineDraw = new MonotonicCubicSpline(PointsXcoordinate, PointsYcoordinate);
SplinePath.add( new scope.Point(UserArray[0].x, UserArray[0].y));
SplineArray.push(new scope.Point(UserArray[0].x, UserArray[0].y));

for (var pointX = UserArray[0].x + interaction_settings.draw_step; pointX <
UserArray[UserArray.length-1].x; pointX = pointX + interaction_settings.draw_step) {
  var pointY = mySplineDraw.interpolate(pointX);
  SplinePath.add( new scope.Point(pointX,pointY));
  SplineArray.push(new scope.Point(pointX,pointY));
}

SplinePath.add( new scope.Point(UserArray[UserArray.length-1].x,
mySplineDraw.interpolate(UserArray[UserArray.length-1].x)));
SplineArray.push(new scope.Point(UserArray[UserArray.length-1].x,
mySplineDraw.interpolate(UserArray[UserArray.length-1].x)));
SplineDrawn.addChild(SplinePath);
}
```

Lastly, the groups are added to the active layer.

```
/* Draw the points and lines of the spline. */
scope.project.activeLayer.addChild(UserCircles);
scope.project.activeLayer.addChild(SplineDrawn);
```


3.6 Creating the response string

For reviewing the code, but more essentially to grade the user response a response string is created. This is done in `create_response()`. The function creates a long, single string which represents an array. The array contains five other arrays:

- | | |
|--------------------------------------|---|
| 1. Spline points in axis coordinates | Used in Grading Code to build a Maple spline |
| 2. User points in pixel coordinates | Used to review answer w.r.t. <code><iframe></code> element. |
| 3. User points in axis coordinates | Used to review answer w.r.t. graph axes. |
| 4. Global maximum coordinates | Used in Grading Code to assess maximum. |
| 5. Global minimum coordinates | Used in Grading Code to assess minimum. |

First, the variables are created:

```
var StringUserPixel = ""; /* string for pixel coordinates by user */
var StringUserAxis = ""; /* string for axis coordinates by user */
var StringSplineAxis = ""; /* string for axis coordinates from spline */
var MinMaxPointsX = []; /* stores spline point axis x-coordinates */
var MinMaxPointsY = []; /* stores spline point axis y-coordinates */
StringResponse = "" /* full string to be evaluated by Möbius */
```

Next, if the length of `UserArray` is non-zero, the strings and the arrays to find the global minimum and maximum are built:

```
for (var i = 0; i < UserArray.length; i++) {
    StringUserPixel = StringUserPixel + "[" + UserArray[i].x + "," + UserArray[i].y +
    "],";
    StringUserAxis = StringUserAxis + "[" + PixelXtoAxisX(UserArray[i].x) + "," +
    PixelYtoAxisY(UserArray[i].y) + "],";
}

for (var i = 0; i < SplineArray.length; i++) {
    StringSplineAxis = StringSplineAxis + "[" + PixelXtoAxisX(SplineArray[i].x) + "," +
    PixelYtoAxisY(SplineArray[i].y) + "],";
    MinMaxPointsX.push(PixelXtoAxisX(SplineArray[i].x));
    MinMaxPointsY.push(PixelYtoAxisY(SplineArray[i].y));
}
```

The ends of the strings are sliced to remove the comma at the end:

```
StringUserPixel = StringUserPixel.slice( 0, -1);
StringUserAxis = StringUserAxis.slice( 0, -1);
StringSplineAxis = StringSplineAxis.slice(0, -1);
```

To find the global maximum and minimum, `MinMaxPointsY` is evaluated for its maximum and minimum. By finding the index of these points, the corresponding x-coordinate in `MinMaxPointsX`. Possibly, the x-coordinate of the global maximum or minimum is outside the x-axis domain. If that is true, a simple fix is done.

Continues on next page.

```
/* Finds spline points with maximum/minimum y-coordinate. */
var max_y = Math.max.apply(null, MinMaxPointsY);
var min_y = Math.min.apply(null, MinMaxPointsY);
/* Finds index of max_y & min_y. */
var val_pos_max = MinMaxPointsY.indexOf(max_y);
var val_pos_min = MinMaxPointsY.indexOf(min_y);
/* Get x-coordinate of max_y & min_y. */
var max_x = MinMaxPointsX[val_pos_max];
var min_x = MinMaxPointsX[val_pos_min];
/* Simple fix if point is outside axes domain (in hpad/vpad area) */
if (min_x < axes[0]) {
    min_x = axes[0];
    min_y = mySplineDraw.interpolate(min_x);
}
if (max_x > axes[1]) {
    max_x = axes[1];
    max_y = mySplineDraw.interpolate(max_x);
}
```

Lastly, the full string is created for StringResponse:

```
StringResponse = "[" + StringSplineAxis + "], [" + StringUserPixel + "], [" +
StringUserAxis + "], [" + String(max_x) + "," + String(max_y) + "]] , [" +
String(min_x) + "," + String(min_y) + "]]]";
```

4 Grading the SketchApp

The `StringResponse` of the user is logged to the webpage console when graded. The log can be accessed by right clicking the webpage and going to *inspect*, or use the keyboard shortcut: `Ctrl+Shift+J` (Windows, Linux), `Command+Option+J` (macOS). If the console is opened before grading, make sure to select *preserve log*.

`StringResponse` is passed to the Möbius **Grading Code** field of the HTML response area as the `$RESPONSE` array. The array elements are the five arrays as described in section 3.6. To access the elements use array indexing, the first element has index 1. Elements 2 and 3 are usually not used when grading a question, but can be useful for error checking coordinate conversion (section 3.1.3).

The code inside the **Grading Code** field is written with Maple code, and thus needs to follow Maple syntax. To start grading, create variables that take the `$RESPONSE` elements and set the initial grade to 0:

```
curv := $RESPONSE[1];
maxi := $RESPONSE[4];
mini := $RESPONSE[5];
grade := 0;
```

Next, the `curv` variable can be used to create Maple spline:

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

To compare the response spline with the correct answer, a simple check can be done by evaluating the spline at the given x-coordinates of the correct answer. A variable, `erro`, stores the deviation of the spline w.r.t. correct answer and is evaluated by an if-statement. In the example below a threshold `erro < 1` is set, but can be anything.

```
erro := 0;
for pt in $answer do erro := erro + abs(pt[2]-(eval(spl, x=pt[1]))) end do;
if erro < 1 then grade := grade+0.25 end if;
```

Also the slopes of the spline can be checked, either to be positive/negative or for a specific value. Below an example where the slope is evaluated at `x = -3`,

```
if is(eval(diff(spl, x), x = -3), positive) then grade := grade+0.25 end if;
if is(abs(eval(diff(spl, x), x = -3)) > 0.25) then grade := grade+0.25 end if;
```

Sometimes, the total area under the spline is important. For instance to check if the net area between -10 and 10 is zero. An area net zero spline is difficult to draw, thus we compare it to the total area. The total area can be estimated from another spline created from the absolute y-values of the original spline:

<code>xvalues := curv[1..nops(curv),1];</code>	List of x-values from curv
<code>yvalues := curv[1..nops(curv),2];</code>	List of y-values from curv
<code>y_array := convert(yvalues, Array);</code>	Convert list into array
<code>y_array_abs := abs(y_array);</code>	Get absolute values
<code>y_array_abs_list := convert(y_array_abs, list);</code>	Convert back to list
<code>L := zip('[]', xvalues, y_array_abs_list);</code>	Combine the two lists
<code>spl_abs := CurveFitting[Spline](L, x);</code>	Fit spline through absolute list
<code>spl_int := abs(int(spl, x = -10 .. 10));</code>	Take integral of original spline
<code>spl_abs_int := abs(int(spl_abs, x = -10 .. 10));</code>	Take integral of absolute spline
<code>ratio_int := spl_int/spl_abs_int;</code>	Calculate the ratio

The variable `ratio_int` is the ratio of the net area of the spline to the total area. The ratio is evaluated with a threshold:

```
if is(ratio_int < 0.2) then grade := grade+0.25;
```

For each criterion discussed above the grade variable, `grade`, is changed. When all criterion are checked grade is returned. The grade is always between 0 and 1, and the weighting of the HTML response area determines the relative grade of the response area for the absolute points of the question (questions can have more than one response area). If the question only consists of the HTML response area and, for instance, the question has a total of 4 points, a grade of 0.5 gives 2 points.

NOTE

The Maple spline needs to be constructed from the SketchApp interpolated output (`$RESPONSE[1]`). If the Maple spline is constructed from the user points in axis coordinates (`$RESPONSE[3]`), the spline will not correspond with SketchApp as shown in Figure 3.

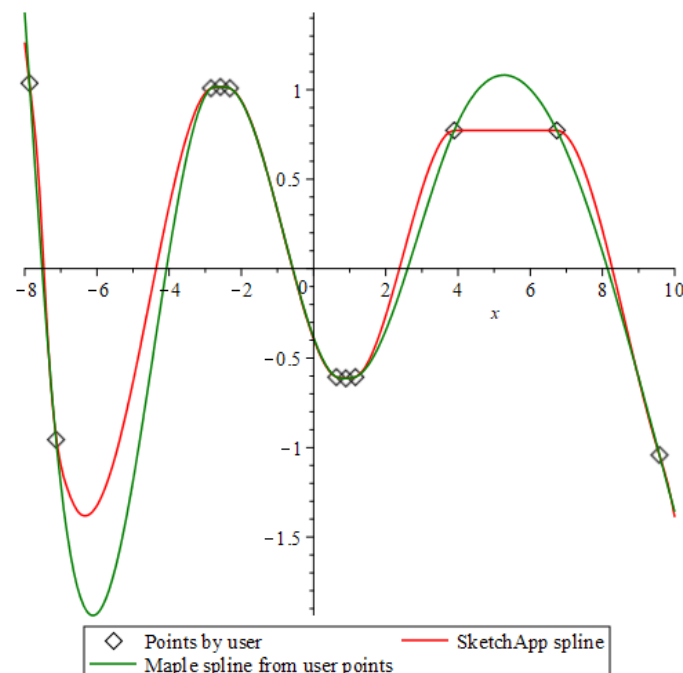


Figure 3: Comparison of SketchApp and Maple spline when only using points by the user.

Appendix A – Algorithm field

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

<code>\$teachermode = "boolean";</code>	Display the HTML table of drawn points
<code>\$answer = "[[x1,y1],[x2,y2],...,[xn,yn]]";</code>	The correct answer, values are floats
<code>\$axes = "[xmin,xmax,ymin,ymax]";</code>	Set the graph axis limits, values are floats
<code>\$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=[``,``])");</code>	This displays the correct answer without the need to preview the question. Updates when save is clicked.
<code>\$canvasDef = "{ width: integer, height: integer, vPad: integer, hPad: integer }";</code>	Set canvas width and height and vertical (vPad) and horizontal (hPad) padding. Values are pixels.
<code>\$backgroundlines = "{ line1: { x: [x1,x2,...,xn], y: [y1,y2,...,yn], lineColor: string, lineColorGreyShade: integer/float, lineThickness: integer } }";</code>	Arrays x and y have float elements. If lineColor is NOT 'grey' then GreyShade = -1, otherwise lineColor is converted to a grey scale. Value for GreyShade between 0 and 1. Add more lines by adding another line2 etc., name can be anything without whitespace.
<code>\$X_axis_definition = "{ Position: string, Flipped: boolean, Arrow: boolean, Name: string, NameJustification: string, NameFontColor: string, NameFontSize: integer, NameHorizontal: integer, NameVertical: integer, NameOrientation: integer, LabelJustification: string, LabelColor: string, LabelFontSize: integer, LabelHorizontal: integer, LabelVertical: integer, LabelNumberPrecision: integer, LabelShowZero: boolean }";</code>	Here all the variables for the X-axis are defined: <ul style="list-style-type: none"> • Position: top, bottom, auto. • NameHorizontal and NameVertical are pixel offset w.r.t. centre of X-axis. • LabelHorizontal and LabelVertical are pixel offset w.r.t. drawn horizontal position and vertical centre of X-axis.
<code>\$Y_axis_definition = "{ Position: string, Flipped: boolean, Arrow: boolean, Name: string, NameJustification: string, NameFontColor: string, NameFontSize: integer, NameHorizontal: integer, NameVertical: integer, NameOrientation: integer, LabelJustification: string, LabelColor: string, LabelFontSize: integer, LabelHorizontal: integer, LabelVertical: integer }";</code>	Here all the variables for the Y-axis are defined: <ul style="list-style-type: none"> • Position: left, right, auto. • NameHorizontal and NameVertical are pixel offset w.r.t. centre of Y-axis. • LabelHorizontal and LabelVertical are pixel offset w.r.t. drawn vertical position and horizontal centre of Y-axis.

<pre> LabelNumberPrecision: integer, LabelShowZero: boolean }"; </pre>	
<pre> \$axes_arrow = "{ AxisLineColor: string, AxisLineThickness: integer, ArrowLineColor: string, ArrowLineThickness: integer, Angle: integer, Size: integer }"; </pre>	<p>If Arrow in X_axis_definition or Y_axis_definition is true, then a line with an arrow head is drawn. AxisLine is the line, ArrowLine is the arrow head line. Angle and Size are related to the arrow head.</p>
<pre> \$major_grid_lines = "{ xStep: float, yStep: float, lineWidth: float, greyScale: float, checkmark_offset: integer, checkmark_color: string, checkmark_width: float }"; </pre>	<p>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).</p>
<pre> \$minor_grid_lines = "{ xStep: float, yStep: float, lineWidth: float, greyScale: float, checkmark_offset: integer, checkmark_color: string, checkmark_width: float }"; </pre>	
<pre> \$interaction_settings = "{ circle_radius: integer, deltax: integer, deltay: integer, draw_step: integer, spline_color: string, spline_width: float }"; </pre>	<p>Define the settings for interaction with the canvas:</p> <ul style="list-style-type: none"> • circle_radius is the size of the user point. • deltax and deltay define the pixel 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 – App behaviour

```

if (type==1) {
    buttons();
    interact();
}
else if (type==2) {
    /* array = "[spline axis] , [user pixel] , [user axis] , [spline axis minimum] ,
[spline axis maximum]" */
    /* spline axis minimum & maximum are only elements of string if getResponse() got
called. */
    /* Each element consists of point coordinates in an array, i.e.
[x1,y1],[x2,y2],...,[xn,yn] */
    var Response = JSON.parse(array);
    var PixelCoor = Response[1];

    for (var i = 0; i < PixelCoor.length; i++) {
        UserArray.push(new scope.Point(PixelCoor[i][0], PixelCoor[i][1]));
    }

    if (gradebook) {
        draw_spline();
    }
    else {
        buttons();
        interact();
        draw_spline();
        create_response();
    }
}
else if (type==3) {
    viewingGradebook = true;
    /* array = "[[x1,y1],[x2,y2],...,[xn,yn]]" as defined by $answer in Möbius Question
Algorithm. */
    var AnswerParsed = JSON.parse(array); /* parses array and returns it without " ". */
    var CorrectAnswer = JSON.parse(AnswerParsed); /* parses array again to get true array.
*/

    for (var i = 0; i < CorrectAnswer.length; i++) {
        UserArray.push(new scope.Point(AxisXtoPixelX(CorrectAnswer[i][0]),
AxisYtoPixelY(CorrectAnswer[i][1])));
    }
    draw_spline();
}
else {
    console.log("Error! incorrect type for RunApp(array,type): type = 1, 2 or 3.");
}

```

Appendix C – Conversion functions

Converting pixel x-coordinate to axis x-coordinate

```
function PixelXtoAxisX(x_Loc){
    if (!x_axis_definition.Flipped) {
        return axes[0] + (x_Loc - canvasDef.hPad) * (major_grid_lines.xStep /
majorX_PixelStep) ;
    }
    else {
        return axes[1] - (x_Loc - canvasDef.hPad) * (major_grid_lines.xStep /
majorX_PixelStep) ;
    }
};
```

Converting pixel y-coordinate to axis y-coordinate

```
function PixelYtoAxisY(y_Loc){
    if (!y_axis_definition.Flipped) {
        return axes[3] - (y_Loc - canvasDef.vPad) * (major_grid_lines.yStep /
majorY_PixelStep) ;
    }
    else {
        return axes[2] + (y_Loc - canvasDef.vPad) * (major_grid_lines.yStep /
majorY_PixelStep) ;
    }
};
```

Converting axis x-coordinate to pixel x-coordinate

```
function AxisXtoPixelX(x_Loc){
    if (!x_axis_definition.Flipped) {
        return canvasDef.hPad + Math.abs((x_Loc - axes[0])) * (majorX_PixelStep /
major_grid_lines.xStep) ;
    }
    else {
        return canvasDef.hPad + Math.abs((x_Loc - axes[1])) * (majorX_PixelStep /
major_grid_lines.xStep) ;
    }
};
```

Converting axis y-coordinate to pixel y-coordinate

```
function AxisYtoPixelY(y_Loc){
    if (!y_axis_definition.Flipped) {
        return canvasDef.vPad + Math.abs((axes[3] - y_Loc)) * (majorY_PixelStep /
major_grid_lines.yStep) ;
    }
    else {
        return canvasDef.vPad + Math.abs((axes[2] - y_Loc)) * (majorY_PixelStep /
major_grid_lines.yStep) ;
    }
};
```


Appendix D – Draw_axis() helper functions

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 = axes_arrow.AxisLineColor;
  axisLine.strokeWidth = axes_arrow.AxisLineThickness ;
  axisLine.add(startPoint);
  axisLine.add(endPoint);
  BackgroundGraph.addChild(axisLine);
  /* add arrow head */
  var vector = endPoint.subtract(startPoint);
  vector.length = axes_arrow.Size;
  var vectorItem = new scope.Path([
    endPoint.add(vector.rotate(axes_arrow.Angle)),
    endPoint,
    endPoint.add(vector.rotate(-axes_arrow.Angle))
  ]);
  vectorItem.strokeWidth = axes_arrow.ArrowLineThickness;
  vectorItem.strokeColor = axes_arrow.ArrowLineColor;
  BackgroundGraph.addChild(vectorItem);
};
```

draw_axis_name()

```
/**
 * Draw the axis name.
 * @param {class} axis_settings - Class from Möbius HTML response area
 * @param {PointText} axistext - Paper.js PointText object
 */
function draw_axis_name(axis_settings, axistext) {
  if (axis_settings.Name != "") {
    axistext.rotate(axis_settings.NameOrientation);
    axistext.justification = axis_settings.NameJustification;
    axistext.fillColor = axis_settings.NameFontColor;
    axistext.fontSize = axis_settings.NameFontSize;
    axistext.content = axis_settings.Name;
    BackgroundGraph.addChild(axistext);
  }
};
```

draw_line()

```
/**
 * Draw a line. Used to draw the major and minor gridlines,
 * and major and minor axis check marks.
 * @param {Integer} x1 - X-coordinate start
 * @param {Integer} y1 - Y-coordinate start
 * @param {Integer} x2 - X-coordinate end
 * @param {Integer} y2 - Y-coordinate end
 * @param {Integer} width - Line width
 * @param {String} colour - Line 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) ;
    BackgroundGraph.addChild(line);
};
```

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 || Math.abs(digit)> 0.00001) {
        text.content = digit.toFixed(NumberPrecision);
    }
    BackgroundGraph.addChild(text);
};
```

pixelBugFix()

```
/**
 * In Chromium browsers (tested: Chrome, MS Edge) gridlines are blurred as they
 * are drawn over 2 pixels.
 * This is fixed by shifting the gridlines.
 * !CAUTION! Fix is unsafe because it checks userAgent string, which is not
 * "unique".
 * @param {Number} number - current pixel
 * @param {Number} value - pixel shift
 * @returns
 */
function pixelBugFix(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 E – Determine axes pixel coordinates

X-axis position

```

if (x_axis_definition.Position != "auto" ) {
    if (x_axis_definition.Position == "top") {
        x_axis_y_coordinate = canvasDef.vPad;
    }
    else if (x_axis_definition.Position == "bottom") {
        x_axis_y_coordinate = canvasDef.height - canvasDef.vPad;
    }
    else {
        console.log("x_axis_position undefined, selector: " +
x_axis_definition.Position + " unknown. [auto, top, bottom]");
    }
}
else {
    if (axes[2] >= 0 && axes[3] > 0 && !y_axis_definition.Flipped) {
        x_axis_y_coordinate = canvasDef.height - canvasDef.vPad;
    }
    else if (axes[2] >= 0 && axes[3] > 0 && y_axis_definition.Flipped) {
        x_axis_y_coordinate = canvasDef.vPad;
    }
    else if (axes[2] < 0 && axes[3] <= 0 && !y_axis_definition.Flipped) {
        x_axis_y_coordinate = canvasDef.vPad;
    }
    else if (axes[2] < 0 && axes[3] <= 0 && y_axis_definition.Flipped) {
        x_axis_y_coordinate = canvasDef.height - canvasDef.vPad;
    }
    else if (!y_axis_definition.Flipped) {
        x_axis_y_coordinate = Math.abs(axes[3])/major_grid_lines.yStep *
majorY_PixelStep + canvasDef.vPad;
    }
    else {
        x_axis_y_coordinate = -Math.abs(axes[3])/major_grid_lines.yStep *
majorY_PixelStep - canvasDef.vPad + canvasDef.height;
    }
}
}

```

Y-axis position

```
if (y_axis_definition.Position != "auto" ) {
    if (y_axis_definition.Position == "left") {
        y_axis_x_coordinate = canvasDef.hPad;
    }
    else if (y_axis_definition.Position == "right") {
        y_axis_x_coordinate = canvasDef.width - canvasDef.hPad;
    }
    else {
        console.log("y_axis_position undefined, selector: " +
y_axis_definition.Position + " unknown. [auto, left, right]");
    }
}
else {
    if (axes[0] >= 0 && axes[1] > 0 && !x_axis_definition.Flipped) {
        y_axis_x_coordinate = canvasDef.hPad;
    }
    else if (axes[0] >= 0 && axes[1] > 0 && x_axis_definition.Flipped) {
        y_axis_x_coordinate = canvasDef.width - canvasDef.hPad;
    }
    else if (axes[0] < 0 && axes[1] <= 0 && !x_axis_definition.Flipped) {
        y_axis_x_coordinate = canvasDef.width - canvasDef.hPad;
    }
    else if (axes[0] < 0 && axes[1] <= 0 && x_axis_definition.Flipped) {
        y_axis_x_coordinate = canvasDef.hPad;
    }
    else if (!x_axis_definition.Flipped) {
        y_axis_x_coordinate = Math.abs(axes[0]) / major_grid_lines.xStep *
majorX_PixelStep + canvasDef.hPad;
    }
    else {
        y_axis_x_coordinate = -Math.abs(axes[0]) / major_grid_lines.xStep *
majorX_PixelStep - canvasDef.hPad + canvasDef.width;
    }
}
```

Appendix F – Drawing the background graph

X-axis line and name

```
if (!x_axis_definition.Flipped) {
    if (x_axis_definition.Arrow) {
        draw_arrow(new scope.Point(canvasDef.hPad, x_axis_y_coordinate),
            new scope.Point(canvasDef.width - canvasDef.hPad + 20, x_axis_y_coordinate));
    }
    var x_temp = pixelBugFix(canvasDef.hPad, 0.5);
}
else {
    if (x_axis_definition.Arrow) {
        draw_arrow(new scope.Point(canvasDef.width - canvasDef.hPad, x_axis_y_coordinate),
            new scope.Point(canvasDef.hPad - 20, x_axis_y_coordinate));
    }
    var x_temp = pixelBugFix(canvasDef.width - canvasDef.hPad, -0.5);
}
var x_axis_text = new scope.PointText(new scope.Point(canvasDef.width/2 + x_axis_definition.NameHorizontal, x_axis_y_coordinate +
x_axis_definition.NameVertical));
draw_axis_name(x_axis_definition, x_axis_text);
```

Y-axis line and name

```
if (!y_axis_definition.Flipped) {
    if (y_axis_definition.Arrow) {
        draw_arrow(new scope.Point(y_axis_x_coordinate, canvasDef.height - canvasDef.vPad),
            new scope.Point(y_axis_x_coordinate, canvasDef.vPad - 20));
    }
    var y_temp = pixelBugFix(canvasDef.height - canvasDef.vPad, 0.5);
}
else {
    if (y_axis_definition.Arrow){
```

```

        draw_arrow(new scope.Point(y_axis_x_coordinate, canvasDef.vPad),
                    new scope.Point(y_axis_x_coordinate, canvasDef.height - canvasDef.vPad + 20));
    }
    var y_temp = pixelBugFix(canvasDef.vPad, 0.5);
}
var y_axis_text = new scope.PointText(new scope.Point(y_axis_x_coordinate + y_axis_definition.NameHorizontal, canvasDef.height/2 +
y_axis_definition.NameVertical));
draw_axis_name(y_axis_definition, y_axis_text);

```

Major vertical gridlines & checkmarks and X-axis labels

```

for (var i = axes[0] ; i <= axes[1] ; i = i + major_grid_lines.xStep) {
    /* major vertical grid line */
    draw_line(x_temp, canvasDef.vPad, x_temp, canvasDef.height - canvasDef.vPad, major_grid_lines.lineWidth, major_grid_lines.lineColor);
    /* major vertical check mark */
    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);
    /* x_axis labels */
    draw_label_text(i, x_temp + x_axis_definition.LabelPositionHorizontal, x_axis_y_coordinate + x_axis_definition.LabelPositionVertical,
        x_axis_definition.LabelJustification, x_axis_definition.LabelColor, x_axis_definition.LabelFontSize, x_axis_definition.LabelShowZero,
        x_axis_definition.LabelNumberPrecision);
    if (!x_axis_definition.Flipped) {
        x_temp = (x_temp + majorX_PixelStep);
    }
    else {
        x_temp = (x_temp - majorX_PixelStep);
    }
}
}

```

Major horizontal gridlines & checkmarks and Y-axis labels

```
for (var i = axes[2] ; i <= axes[3] ; i = i + major_grid_lines.yStep) {  
    /* major horizontal grid line */  
    draw_line(canvasDef.hPad, y_temp, canvasDef.width - canvasDef.hPad, y_temp, major_grid_lines.lineWidth, major_grid_lines.lineColor);  
    /* major horizontal check mark */  
    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);  
    /* y_axis labels */  
    draw_label_text(i, y_axis_x_coordinate + y_axis_definition.LabelPositionHorizontal, y_temp + y_axis_definition.LabelPositionVertical,  
        y_axis_definition.LabelJustification, y_axis_definition.LabelColor, y_axis_definition.LabelFontSize, y_axis_definition.LabelShowZero,  
        y_axis_definition.LabelNumberPrecision);  
    if (!y_axis_definition.Flipped) {  
        y_temp = y_temp - majorY_PixelStep;  
    }  
    else {  
        y_temp = y_temp + majorY_PixelStep;  
    }  
}
```

Minor vertical gridlines & check marks

```
x_temp = pixelBugFix(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 check mark (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_PixelStep;  
}
```


Minor horizontal gridlines & check marks

```
y_temp = pixelBugFix(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 check mark (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_PixelStep;
}
```

Appendix G – Drawing the background line

The background spline is drawn similarly to the user spline, see section 3.5. However, since the monotonic cubic spline is created in axis coordinates a conversion to pixel coordinates is needed. Also, drawing the background spline can be limited to a certain range. This is defined by the `x_limit_min` and `x_limit_max` properties.

```
for (var property in backgroundlines) {
    if (backgroundlines.hasOwnProperty(property)) {
        var lineObject = backgroundlines[property];
        var x_val = lineObject.x;
        var y_val = lineObject.y;
        var mySplineGraph = new MonotonicCubicSpline(x_val, y_val);
        var SplineGraphPath = new scope.Path();
        SplineGraphPath.strokeWidth = backgroundlines.lineThickness;

        if (lineObject.lineColorGreyShade < 0) {
            SplineGraphPath.strokeColor = lineObject.lineColor;
        }
        else {
            SplineGraphPath.strokeColor = new scope.Color(lineObject.lineColorGreyShade);
        }

        for (var pointX = AxisXtoPixelX(x_val[0]); pointX <= AxisXtoPixelX(x_val[x_val.length-1]); pointX = pointX +
interaction_settings.draw_step) {
            var pointY = mySplineGraph.interpolate(PixelXtoAxisX(pointX));
            if ((!(lineObject.hasOwnProperty("x_limit_max")) || ((lineObject.hasOwnProperty("x_limit_max")) &&
AxisXtoPixelX(lineObject.x_limit_max) >= pointX)) && (!(lineObject.hasOwnProperty("x_limit_min")) ||
((lineObject.hasOwnProperty("x_limit_min")) && AxisXtoPixelX(lineObject.x_limit_min) <= pointX))) {
                SplineGraphPath.add(new scope.Point(pointX, AxisYtoPixelY(pointY)));
            }
        }
    }
}
```

```
        if (!(lineObject.hasOwnProperty("x_limit_max")) || ((lineObject.hasOwnProperty("x_limit_max")) &&
(lineObject.x_limit_max >= x_val[x_val.length-1]))) {
            SplineGraphPath.add(new scope.Point(AxisXtoPixelX(x_val[x_val.length-1]), AxisYtoPixelY(y_val[x_val.length-1])));
        }
        BackgroundGraph.addChild(SplineGraphPath);
    }
}
```

Appendix H – Mouse interaction

```
function interact() {
  tool.onMouseDown = function(click) {
    hitPoint = click.point;
    var hitResult = UserCircles.hitTest(hitPoint, hitOptions);
    if (!hitResult) {
      /* deselects point */
      if (selected_x !== null || selected_y !== null) {
        selected_x = null;
        selected_y = null;
      }
      else {
        if (UserArray.length == 0) {
          UserArray.push(hitPoint); /* first point */
        }
        else {
          for (var i = 0 ; i < UserArray.length ; i++) {
            if (hitPoint.x < UserArray[i].x) {
              UserArray.splice(i, 0, hitPoint); /* insert point */
              break;
            }
            else if (i == UserArray.length - 1) {
              UserArray.splice(UserArray.length, 0, hitPoint); /*
append to end */
              break;
            }
          }
        }
      }
    }
    else {
      selected_x = hitResult.item.position.x;
      selected_y = hitResult.item.position.y;
    }
    draw_spline();
    create_response();
  };

  tool.onMouseDrag = function(click) {
    var mouseStartLocation = click.point;
    var mouseMovedDistance = click.delta;
    var results = [];
    for (var i = 0 ; i < UserArray.length ; i++) {
      /* Push all points at mouse location */
      if (mouseStartLocation.getDistance(UserArray[i]) <
hitOptions.tolerance) {
        results.push(i);
      }
    }
  }
}
```

```
    }  
    /* Remove existing point by splicing */  
    for (var i = (results.length-1) ; i >= 1 ; i--) {  
        UserArray.splice(results[i], 1);  
    }  
  
    UserArray[results[0]] = new scope.Point(mouseStartLocation.x +  
mouseMovedDistance.x, mouseStartLocation.y + mouseMovedDistance.y);  
    draw_spline();  
    create_response();  
};  
}
```

Appendix I – Button functionality

```
function buttons(){
  var delPoint = $('#delPoint'); /* Delete the selected point */
  var delAll = $('#delAll'); /* Delete all points */
  var localMin = $('#localMin'); /* Force the selected point to be a local minimum */
  var localMax = $('#localMax'); /* 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 */

  delPoint.click(function() {
    for ( var i = 0 ; i < UserArray.length ; i++ ) {
      if ((Math.abs(UserArray[i].x - selected_x) < hitOptions.tolerance) &&
(Math.abs(UserArray[i].y - selected_y) < hitOptions.tolerance)) {
        UserArray.splice(i, 1);
      }
    }
    selected_x = null;
    selected_y = null;
    draw_spline();
    create_response();
  });

  delAll.click(function() {
    UserArray.splice(0, UserArray.length);
    UserCircles.removeChildren();
    SplineDrawn.removeChildren();
    selected_x = null;
    selected_y = null;
    create_response();
  });

  localMin.click(function() {
    for ( var i = 0 ; i < UserArray.length ; i++ ) {
      if ((Math.abs(UserArray[i].x - selected_x) < hitOptions.tolerance) &&
(Math.abs(UserArray[i].y - selected_y) < hitOptions.tolerance)) {
        UserArray.splice(i+1, 0, new scope.Point(UserArray[i].x +
interaction_settings.deltax, UserArray[i].y - interaction_settings.deltay));
        UserArray.splice(i,0, new scope.Point(UserArray[i].x -
interaction_settings.deltax, UserArray[i].y - interaction_settings.deltay));
        break;
      }
    }
    selected_x = null;
    selected_y = null;
    draw_spline();
    create_response();
  });
}
```

```
localMax.click(function() {
    for ( var i = 0 ; i < UserArray.length ; i++ ) {
        if ((Math.abs(UserArray[i].x - selected_x) < hitOptions.tolerance) &&
(Math.abs(UserArray[i].y - selected_y) < hitOptions.tolerance)) {
            UserArray.splice(i+1, 0, new scope.Point(UserArray[i].x +
interaction_settings.deltax, UserArray[i].y + interaction_settings.deltay));
            UserArray.splice(i,0, new scope.Point(UserArray[i].x -
interaction_settings.deltax, UserArray[i].y + interaction_settings.deltay));
            break;
        }
    }
    selected_x = null;
    selected_y = null;
    draw_spline();
    create_response();
});

toggleContrast.click(function() {
    var div_contrast = document.getElementById('contrast');
    if (div_contrast.style.display === 'none') {
        div_contrast.style.display = 'block';
    }
    else {
        div_contrast.style.display = 'none';
    }
});

gridMajor.click(function() {
    BackgroundGraph.removeChildren();
    major_grid_lines.lineColor = gridMajor.val()/10;
    draw_axis();
});

gridMinor.click(function() {
    BackgroundGraph.removeChildren();
    minor_grid_lines.lineColor = gridMinor.val()/10;
    draw_axis();
});
});
```