Workshop: Advanced JSXGraph

1. International JSXGraph Conference

Alfred Wassermann

UNIVERSITÄT
BAYREUTH

06/07-10-2020

Contents

PART ONE

Preliminaries

Include JSXGraph

- JSXGraph skeleton page:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>JSXGraph template</title>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <link href="https://cdn.jsdelivr.net/npm/jsxgraph@1.1.0/distrib/
        jsxgraph.css" rel="stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph@1.1.0/distrib/
        jsxgraphcore.js" type="text/javascript" charset="UTF-8"></script>

    <!-- The next line is optional: MathJax -->
    <script src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js"
        id="MathJax-script" async></script>
  </head>
  <body>

  <div id="jxgbox" class="jxgbox" style="width:500px; height:200px;"></div
      >

  <script>
    var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-5, 2, 5,
        -2]});
  </script>

  </body>
</html>
```

- See JSXGraph handbook (in development): https://ipesek.github.io/jsxgraphbook/

Functions in JavaScript

- JavaScript function:

```
var f = function(x) {
  return x * x;
};
```

- JavaScript arrow functions:

```
var f = (x) => x * x;
```

- Use of anonymous functions:

```
myfunction(x, function(t) { return t * t; });
myfunction(x, (t) => t * t);
```

Function plotting in JSXGraph

```
board.create('functiongraph', [function(x) { return x * x; }]);
board.create('functiongraph', [(x) => Math.sin(x)]);
board.create('functiongraph', ['sin(x) + 1']);
```

- See https://jsxgraph.org/wiki/index.php/Category:Curves

Dependent / dynamic elements

- Javascript functions can be used to connect JSXGraph elements and make one element dependent on another. After each `move` event all coordinates of all elements are updated.
- See https://jsfiddle.net/yjg5kruL/
- This works for all coordinates in JSXGraph

```
var free = board.create('point',[0,0], {name:'A', size:3});

var dep1 = board.create('point',[
      function(){ return free.X();},
      1
    ], {name:'B', face:'[]', size:3});

var dep2 = board.create('point',[
      () => free.X(),
      () => free.Y() + 2
    ], {name:'C', face:'[]', size:3});
```

- Function plots with sliders: https://jsxgraph.org/wiki/index.php/Slider_and_function_plot

Dynamic attributes

- The same approach works also for nearly all *attributes* of JSXGraph elements

- See https://jsfiddle.net/dgqk7epy/

```
var slider = board.create('slider', [[-3, 4], [3,4], [0, 0.3, 1]], {
               name: "opacity"});

var q = board.create('regularpolygon', [[-1,-1],[1,-1],4], {
           fillColor:'yellow', hasInnerPoints: true,
           fillOpacity: function() { return slider.Value(); },
           visible: () => (slider.Value() == 0) ? false : true,
           vertices: {
                   visible: () => (slider.Value() == 0) ? false : true,
           }
        });
```

Setting attribute values

- There various methods to set attributes for JSXGraph elements.
- An attribute can be set *globally* for the whole page by setting e.g. `JXG.Options.point.color` **before** initialising the first board.
- The attribute is set globally for the board by setting e.g. `board.options.point.size = 10;` after the call of `board = JXG.JSXGraph.initBoard();`.
- An attribute can be set for an individual element by supplying it in the attribute object parameter: `var p3 = board.create('point', [1,3], {size: 15});`
- An attribute can be changed with e.g. `p3.setAttribute({color: 'yellow'});`
- See https://jsfiddle.net/62uftrmo/2/

```
// Global to the page
JXG.Options.point.color = 'blue';
JXG.Options.point.size = 5;

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis:true
});

var p1 = board.create('point', [1,1]);

// Global to the board
board.options.point.size = 10;

var p2 = board.create('point', [1,2]);

// Local
var p3 = board.create('point', [1,3], {size: 15});

// Later:
```

```
p3.setAttribute({color: 'yellow'});
```

Include MathJax

- MathJax: Beautiful and accessible math in all browsers
- Allows LaTeX syntax
- In JSXGraph: backslash \ has to be replaced by \\
- Can be used in dynamic texts
- Include MathJax: add MathJax JavaScript code

```
    <script src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js"
       id="MathJax-script" async></script>
```

- JSXGraph: set attribute `useMathJax` for text elements
- See https://jsfiddle.net/uvp3mcf7/
- See also https://jsxgraph.org/wiki/index.php/Matrix_multiplication_II

```
JXG.Options.text.useMathJax = true;
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 20, 5, -5], axis:true
});

var k = board.create('slider',[[-4,-2],[3,-2],[-5,1,5]],
            {name:'n', snapWidth:1});

board.create('functiongraph', [
    function(t) {return JXG.Math.pow(Math.E,t*k.Value());}
 ],
 {strokeColor:'#ff0000'});

// LaTeX: \[ e^{3x} \]
// MathJax text:
board.create('text',[-4, 7,
  function() {
    return '\\[f(x) = e^{' + k.Value() + 'x}\\]';
  }],
  {fontSize: 24});
```

Include HTML code

- The JSXGraph text element allows to include arbitrary HTML code.
- Example: https://jsxgraph.org/wiki/index.php/Self-contained_function_plotting

- Exception: axis tick labels. Their default attribute is `display: 'internal'` and they are SVG text elements.
- Example (https://jsfiddle.net/8d1byuft/):

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis:true
});

board.create('text', [1, 3,
    '<h1>This is HTML</h1><input type="checkbox"> OK<br><input type="
        checkbox"> not OK<br>'])
```

Bidirectional communication with DOM

- For some applications it is useful to
  - show some value of a JSXGraph element outside in the outside HTML text
  - and allow to set this value by changing the HTML text.
- Example: https://jsxgraph.org/wiki/index.php/Bearing
- Solution in that example:
  - Use an event listener for the `drag` event for this element which does the writing to HTML
  - Call `moveTo` from “outside” to move the element.

PART TWO

Vector symbols

```
board.create('point', [2, 1], {name: 'v&#8407;'})

var vec = '<math xmlns="http://www.w3.org/1998/Math/MathML"><mover><mi>v</
    mi><mo>&rarr;</mo></mover></math>';

board.create('point', [2, 2], {name: vec});
```

- See https://jsfiddle.net/m0u8Love/

Building a theme

- Individual properties of an attribute object can be added / changed later on:

```
var mypoint = {
        color: 'blue',
        highlightFillColor: 'yellow',
        highlightStrokeColor: 'yellow',
        size: 5
    };
var p1 = board.create('point', [1,2], mypoint);

mypoint.opacity = 0.3;
// Alternatively:
mypoint['opacity'] = 0.3;

var p2 = board.create('point', [1,3], mypoint);
```

- See https://jsfiddle.net/2quxtv38/
- To add own settings for objects to the default JSXGraph options, *merge* the new settings into the `JXG.Options` object before the first call of `initBoard`:

```
JXG.Options = JXG.merge(JXG.Options, {
    point: {
        color: 'blue',
        highlightFillColor: 'yellow',
        highlightStrokeColor: 'yellow',
        size: 5
    }
});

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis:true
});

var p1 = board.create('point', [1,2]);
```

- See https://jsfiddle.net/2quxtv38/1/
- See https://github.com/jsxgraph/jsxgraph/blob/master/src/options.js for the definition of the default attributes
- See https://github.com/jsxgraph/jsxgraph/blob/master/src/themes/dark.js for an example theme.

Iterate over all objects

- `board.objects`: key-value store (object)
- `board.objectsList`: array with chronological order
- Select elements `board.select()`
- See https://jsxgraph.org/docs/symbols/JXG.Board.html
- Each JSXGraph has the properties `type`, `elementClass` and `elType`, see https://github.com/jsxgraph/jsxgraph/blob/master/src/base/constants.js

Performance

- `create` and `remove` is expensive. Use `visible:true/false` if possible.
- Use `withLabel:false` for points

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    axis:true,
    boundingbox:[-0.1, 1.1, 1.1, -0.1]
});

var N = 2000;
var points = [];

console.time("create");
for (let i = 0; i < N; i++) {
    points.push(board.create('point', [Math.random(), Math.random()], {
        withLabel:false}));
}
console.timeEnd("create");

console.time("remove");
board.suspendUpdate();
for (let i = 0; i < N; i++) {
    board.removeObject(points[i]);
}
board.unsuspendUpdate();
console.timeEnd("remove");
```

- Advanced function plotting: enhance performance by doing expensive initialisation only at the first evaluation during an `update` call.

```
var f = function(x, suspendedUpdate) {
    if (!suspendedUpdate) {
        // Do some expensive initial calculation only for the first
           evaluation
    }
    return value /* depending on x */;
};
var plot = board.create('functiongraph', [f]);
```

- Example: see https://github.com/jsxgraph/jsxgraph/blob/master/src/element/conic.js
- Random points: see also https://jsxgraph.org/wiki/index.php/Random_points
- Random walks: see https://jsxgraph.org/wiki/index.php/Random_walks

Constructing with loops

- Constructing multiple JSXGraph elements in a JavaScript loop provides some pitfalls if the loop counter variable is used in a function to define the JSXGraph elements.
- Example with arrow functions (traditional functions have the same problem):

```
var s = board.create('slider', [[1,4],[4,4], [0,1,5]]);
// Does not work
var j;
for (j = 0; j < 4; j++) {
    board.create('segment', [
      [j, 0],
      [j, () => -j * s.Value()]
  ]);
}
```

- The solution are `closures`, i.e. a function that returns the required function:

```
// Solution with closures
var j;
for (j = 0; j < 4; j++) {
    board.create('segment', [
      [-j, 0],
      [-j, (function(jj) { return () => -jj * s.Value(); })(j)]
  ]);
}
```

- Fortunately, by defining the loop counter variable with `let` instead of `var` our example works out of the box:

```

// Works
for (let i = 0; i < 4; i++) {
    board.create('segment', [
      [i, 0],
      [i, () => i * s.Value()]
```

```
  ]);
}
```

- See https://jsfiddle.net/3L4y52w9/

JessieCode

- JessieCode is a programming language developed by Michael Gerhäuser (talk on Thursday) which creates JSXGraph elements or allows easier function plots.
- Reference: https://bin.sketchometry.com/ref
- Examples:
  - https://bin.sketchometry.org
  - https://jsxgraph.org/wiki/index.php/Euler_line_source_code
  - https://jsxgraph.org/wiki/index.php/Shade_region_bounded_by_curves
  - https://jsxgraph.org/wiki/index.php/Even_simpler_function_plotter
  - Function plot: https://jsfiddle.net/2px0wryb/
  - JessieCode tag: https://jsfiddle.net/5nz9em86/

```
var plot = board.create('functiongraph', ['sin(x)']);
var f = board.jc.snippet('sin(x)', true, 'x');
console.log(f(Math.PI / 2));
```

- See also: https://jsxgraph.org/docs/symbols/JXG.JessieCode.html#snippet

Dump a construction

- JSXGraph provides several methods to save the state of a construction:
  - as image (data URI or canvas image)
  - as JavaScript code
  - as JessieCode code
- The dumped data can be used to store students results on a server.
- See https://jsfiddle.net/4ua2Lhfc/1/

```
var p = board.create('point', [Math.PI/2,1], {name: 'A'});
var f = board.create('functiongraph', ['A.Y() * sin(x)']);

function dump() {
  var txt;
  // dataURI:
  txt = board.renderer.dumpToDataURI();
  // JavaScript:
  txt = JXG.Dump.toJavaScript(board);
  // JessieCode
  txt = JXG.Dump.toJessie(board);

  document.getElementById('output').value = txt;
}
```

Homework: performance

- **Task:** Highlight the area whose points are in a certain relation to given, draggable points.
- **Solution 1:** Use many points and optimize the update operation, see https://jsfiddle.net/o08au37y

```
var N = 100;
var radius = 20.2;

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [0, N, N, 0], axis:false
});

var points = [];

var p = board.create('point', [N/2, N/2], {
                        color:'blue',
            highlightStrokeColor: 'blue',
            highlightFillColor: 'blue',
            size: 10,
            name: ''
          });

board.suspendUpdate();
for (let i = 0; i < N; i++) {
  for (let j = 0; j < N; j++) {
    points.push(board.create('point', [i, j], {
      size: 1,
      withLabel: false,
      fixed: true,
      layer: 1
    }));
    points[i * N + j].setAttribute({
      visible: function() { return points[i * N + j].Dist(p) < radius; }
    });

    // Disable highlighting
    points[i * N + j].hasPoint = function() {};
  }
}
board.unsuspendUpdate();
```

- **Solution 2:** Use a single curve which may cover the whole board. See https://jsfiddle.net/o08au37y/5. Some comments on the code:
  - In `curve.updateDataArray` the coordinates of the points of the path are set by using the arrays `this.dataX` and `this.dataY`
  - Adding `NaN` interrupts the curve path.
  - `board.update()` has to be called after setting `curve.updateDataArray`

```
var N = 200;
var radius = 20.2;

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [0, N, N, 0], axis:true
});

var points = [];

var p = board.create('point', [N/2, N/2], {
                        color:'blue',
            highlightStrokeColor: 'blue',
            highlightFillColor: 'blue',
            size: 10,
            name: ''
          });

var area = board.create('curve', [[4, 5], [3, 2]], {strokeWidth: 1,
    strokeColor: 'red'});
area.hasPoint = function() {};
area.updateDataArray = function() {
  var dx, dy,
        r = radius * radius;

  this.dataX = [];
  this.dataY = [];

  for (let i = 0; i < N; i++) {
    dx = i - p.X();
    dx *= dx;
    for (let j = 0; j < N; j++) {
      dy = j - p.Y();
      dy *= dy;

      if (dx + dy < r) {
        this.dataX.push(i);
        this.dataX.push(i);
        this.dataY.push(j);
        this.dataY.push(j + 1);
      }
    }
    this.dataX.push(NaN);
    this.dataY.push(NaN);
  }
};
board.update();
```