Workshop: Advanced JSXGraph

Vol. 5

Alfred Wassermann

UNIVERSITÄT BAYREUTH

24-03-2021

Contents

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@1.2.2/distrib/jsxgraph.css" rel="stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph@1.2.2/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/

New in version 1.2.2

Clipping: intersection, union, difference

- In version 1.2.2 clipping of curves (intersection, union, difference) has been much improved. Also arcs and sectors can now be intersected.
- Example: create a gauge by clipping out a circle from a sector. This example has been inspired by Matti Hurjala.
- https://jsfiddle.net/47srb0qa/9/

```
var p1 = board.create('point',[0, 0], {visible: false});
var p2 = board.create('point',[1, 0], {visible: false});

var c1 = board.create('circle', [p1, 1], {visible: false});
var p3 = board.create('glider',[-1, 1, c1], {
          visible: true,
          name:'',
          showInfobox: false
        });

var sector = board.create('sector', [p1, p2, p3], {visible: false});
var circle = board.create('circle', [p1, 0.5], {visible: false});

var curve = board.create('curve', [[],[]], {
            fillColor: 'blue',
            fillOpacity: 0.5
          });

curve.updateDataArray = function() {
    var a = JXG.Math.Clip.difference(sector, circle, this.board);
    this.dataX = a[0];
    this.dataY = a[1];
};

var txt = board.create('text', [1, 1,
       () => (50 * JXG.Math.Geometry.rad(p2, p1, p3) / Math.PI).toFixed(0)
           + ' %'
    ]);
board.update();
```

Moreover, intersecting intersections should be possible now, see https://jsfiddle.net/x2ywfL03/.

```
JXG.Options.label.autoPosition = true;
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-5, 5, 5, -5], axis:true
});
var p1 = board.create('point',[0, 0]);
var p2 = board.create('point',[3, 0]);
var p3 = board.create('point',[-3,-3]);

var circle1 = board.create('circle', [p1, 2.0], {visible: false});
var circle2 = board.create('circle', [p1, 0.5], {visible: false});

var ring = board.create('curve', [[],[]], {
             fillColor: 'blue',
             fillOpacity: 0.1
           });
ring.updateDataArray = function() {
    var a = JXG.Math.Clip.difference(circle1, circle2, this.board);
    this.dataX = a[0];
```

```
    this.dataY = a[1];
};
board.update();

var poly = board.create('polygon', [p1, p2, p3], {fillColor: 'red'});

var curve1 = board.create('curve', [[],[]], {
                fillColor:'yellow',
                strokeWidth:2,
                fillOpacity: 0.8
              });
curve1.updateDataArray = function() {
    var a = JXG.Math.Clip.intersection(ring, poly, this.board);
    this.dataX = a[0];
    this.dataY = a[1];
};
board.update();
```

Lesser known elements

Let's have a look at some JSXGraph features which are not so well known.

Segments of fixed length

If the constructor of a segment has a third parameter, this parameter fixes the length of the segment. Of course, the defining points have to allow this. As a result, if one end point of the segment is dragged, the behaviour of the "other" point is somewhat unpredictable following the dragged point. See https://jsfiddle.net/9yceb56w/2/.

```
JXG.Options.label.autoPosition = true;
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-1, 5, 5, -1], axis:true
});

var A = board.create('point', [1.0, 1.0]);
var B = board.create('point', [3.0, 1.0]);
// Segment AB is a regular / unrestricted segment
var l1 = board.create('segment', [A, B]);

var C = board.create('point', [1.0, 2.0]);
var D = board.create('point', [3.0, 2.0]);
// Segment CD has fixed length 2
var l2 = board.create('segment', [C, D, 2]);

var E = board.create('point', [1.0, 3.0]);
```

```
var F = board.create('point', [3.0, 4.0]);
// Segment EF has the same length as AB
var l3 = board.create('segment', [E, F, () => l1.L() ]);
```

Fixed angles

If the defining third point allows it, an angle can be set to a fixed value or a function setting the angle value.

- Fixed value, see https://jsfiddle.net/avqjpghc/2/

```
var p1, p2, p3, a;

p1 = board.create('point',[0, 0]);
p2 = board.create('point',[3, 0]);
p3 = board.create('point',[0, 3]);

board.create('line', [p1, p2], {strokeWidth: 1});
board.create('line', [p1, p3], {strokeWidth: 1});

a = board.create('angle', [p2, p1, p3], {radius: 1});
a.setAngle(Math.PI / 3);
board.update();
```

```
<button onClick="a.free()">
Free the angle
</button>
<button onClick="a.setAngle(Math.PI / 3); board.update();">
Fix the angle
</button>
```

- Dynamic value, see https://jsfiddle.net/xcLupj24/2/

```
var p1, p2, p3, a, s;

s = board.create('slider', [[0.5, -2], [3.5, -2], [0, Math.PI/3, Math.PI
    ]], {
      name: '&gamma;'
    });
p1 = board.create('point',[0, 0], {name: ''});
p2 = board.create('point',[3, 0]);
p3 = board.create('point',[0, 3]);

board.create('line', [p1, p2], {strokeWidth: 1});
board.create('line', [p1, p3], {strokeWidth: 1});

a = board.create('angle', [p2, p1, p3], {radius: 0.5, name:"&gamma;"});
a.setAngle(() => s.Value());
```

```
board.update();
```

Magnetized points

It is possible to bind points “a little bit” to other objects: if such a “magnetized” point is closer than the `attractorDistance` to one of the attracting elements, the point mutates into a glider object on that attracting element. Only if there is a rapid drag on that point which is larger than `snatchDistance` away from the attracting element, the point is set free again. See https://jsfiddle.net/vanmtejq/3/.

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

// Elements to which act magnetic to point p
var li = board.create('line', [1,2,3]),
    ci = board.create('circle', [[2,2],1]),
    p0 = board.create('point', [-2,0], {color:'blue'});

// A magnetized point:
var p = board.create('point', [-2,-2], {
        attractors: [li, ci, p0],
        attractorDistance:0.2,
        snatchDistance: 2
    });
```

- Short form: use `snapToPoints:true` and set `attractorDistance` and `snatchDistance`, see https://jsfiddle.net/xud5gvzo/1/.

```
JXG.Options.line.point1.visible = true;
JXG.Options.line.point2.visible = true;
JXG.Options.line.point1.color = 'red';
JXG.Options.line.point2.color = 'red';
JXG.Options.line.point1.size = 5;
JXG.Options.line.point2.size = 5;
JXG.Options.line.point1.layer = 9;
JXG.Options.line.point2.layer = 9;

JXG.Options.point.snapToPoints = true;
JXG.Options.point.attractorDistance = 0.5;
JXG.Options.point.snatchDistance = 0.5;

const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-1, 5, 5, -1], axis:true
});

var s1 = board.create('segment', [[0, 1], [1.5, 1]]);
```

```
var s2 = board.create('segment', [[0, 2], [1.5, 2]]);
var s3 = board.create('segment', [[0, 3], [1.5, 3]]);
var s4 = board.create('segment', [[0, 4], [1.5, 4]]);
```

The power of turtle graphics

"In computer graphics, *turtle graphics* are vector graphics using a *relative cursor* (the 'turtle') upon a Cartesian plane. Turtle graphics is a key feature of the *Logo* programming language" (https://en.wikipedia.org/wiki/Turtle_graphics, https://www.cse.wustl.edu/~taoju/research/TurtlesforCADRevised.pdf)

PostScript and SVG may also be regarded as turtle graphics. In JSXGraph, turtle elements transfer directly to SVG path elements, i.e. are handled quite efficiently.

- A first example, see https://jsfiddle.net/s5ujapmr/6/

```
var t = board.create('turtle', [10, 10, 0], {arrow: {visible: false}});

t.forward(10);
t.right(30);
t.forward(10);

t.setPos(-10, 10);

for (let i = 0; i < 10; i++) {
  t.forward(5);
  t.right(36);
}

t.penUp();
t.moveTo([20, 30]);
t.penDown();
t.setPenColor('red');
t.setPenSize(3);
t.lookTo([10, 0]);
t.fd(10).lt(90).fd(10).lt(90).fd(10).lt(90).fd(10);
t.setAttribute({fillColor:'yellow'});
```

Available commands

For the turtle object, angles have to be supplied in degrees, not in radians.

- `t.forward(len);` or `t.fd(len);`
- `t.back(len);` or `t.bk(len);`
- `t.right(angle);` or `t.rt(angle);` $(0 \leq \alpha \leq 360)$

- `t.left(angle);` or `t.lt(angle);` $(0 \leq \alpha \leq 360)$
- `t.penUp();` or `t.pu();`
- `t.penDown();` or `t.pd();`
- `t.clearScreen();` or `t.cs();`
- `t.clean();`
- `t.setPos(x,y);`
- `t.home();`
- `t.hideTurtle();` or `t.ht();`
- `t.showTurtle();` or `t.st();`
- `t.setPenSize(size);` (size: number)
- `t.setPenColor(col);` (col: colorString, e.g. “red” or “#ff0000”)
- `t.setAttribute({key1:value1,key2:value2,...});`
- `t.pushTurtle()`
- `t.popTurtle()`
- `t.lookTo(dir)` or `t.lookTo([x,y])` (Turtle looks to a coordinate pair. If `t2` is another turtle object: `t.lookTo(t2.pos)`)
- `t.moveTo([x,y])` (Turtle goes forward to the coordinates pair. The turtles direction is not changed.)
- `t.X()`, `t.Y()`: get the position of the turtle `t`.

More possibilities

- Animated function graph drawing, see https://jsfiddle.net/7sthap9g/2/

```
var f = function(x) { return Math.sin(x); };

var start = -4,
    end = 4,
    x = start,
    step = 0.2,
    turtle = board.create('turtle', [x, f(x)], {fixed: false});

turtle.hideTurtle();     // Hide the turtle arrow

var moveForward = function() {
        x += step;
        turtle.moveTo([x, f(x)]);
        if (x >= end) {
           return;
        }
        setTimeout(moveForward, 200); // delay by 200 ms
     };
```

```
moveForward();              // Start the drawing
```

- Fractals, see https://jsfiddle.net/m4crjns1/1/

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-300, 300, 300, -300], axis: false});


var edge = function(size, level) {
    if (level ==0 ) {
        t.fd(size);
        return;
    }
    edge(size / 3, level - 1);
    t.lt(60);
    edge(size / 3, level - 1);
    t.rt(120);
    edge(size / 3, level - 1);
    t.lt(60);
    edge(size / 3, level - 1);
};

var snowflake = function(size, level) {
    var i;
    for (i = 0; i < 3; i++) {
        edge(size, level);
        t.rt(120);
    };
}

var t = board.create('turtle');
t.clearScreen();
t.hideTurtle();
t.setPenSize(1)
t.setPenColor("#000000")
t.lt(30);
t.setPos(0,-100);
snowflake(250, 4);
```

See more examples of fractals at https://jsxgraph.org/wiki/index.php/Category:Fractals

Turtle graphics and differential equations

Let's look at the following situation:

There is a population y which consists at time t_0 of s individuals (or units) and which grows in each time span Δt by a factor α of its actual size. Therefore:

$$\Delta y = \alpha \cdot y \cdot \Delta t \quad \Rightarrow \quad \frac{\Delta y}{\Delta t} = \alpha \cdot y.$$

If the time span gets infinitely small, the growth of the population becomes

$$y' = \frac{dy}{dt} = \alpha \cdot y$$

Such an equation is called a *differential equation*. The solutions of differential equations are functions. Some differential equations can be solved analytically, as is the case with our equation from above which has as solution the function $y(t) = se^{\alpha t}$ (compute the derivative and check).

A graphical solution of a such a differential equation can be approximated easily by *turtle graphics* (which results in effect to Euler's method), see https://jsfiddle.net/un6yvdxk/11/.

```
const board = JXG.JSXGraph.initBoard('jxgbox', {
    boundingbox: [-0.25, 12.5, 20, -12.5], axis: true
});

var s     = 0.1;
var alpha = 0.3;
var dt    = 0.1;  // step width delta t, global variable

// Correct solution
var solution = board.create('functiongraph', [
          (x) => s * Math.exp(alpha * x)
    ], {strokeColor:'red'});

// Approximation by the turtle
var turtle = board.create('turtle');
turtle.hideTurtle();

turtle.setPenSize(4);
turtle.setPos(0, s);

var loop = function() {
  var dy = alpha * turtle.Y() * dt;   // Increase of y
  turtle.moveTo([dt + turtle.X(), dy + turtle.Y()]);
  if (turtle.X() < 20.0) {
     setTimeout(loop, 5);
  }
};

loop();
```

See also

- Abelson, diSessa: Turtle Geometry - The Computer as a Medium for Exploring Mathematics

- https://ars.me/curiosities/2014/06/25/lorenz-attractor-turtle/
- https://jsxgraph.org/wiki/index.php/Population_growth_models
- https://jsxgraph.org/wiki/index.php/Epidemiology:_The_SIR_model

More examples

- https://jsxgraph.org/wiki/index.php/Turtle_Graphics
- https://jsxgraph.org/wiki/index.php/Programming_turtle_graphics
- https://jsxgraph.org/wiki/index.php/Random_walks
- https://jsxgraph.org/wiki/index.php/Turtle_bot
- https://jsxgraph.org/wiki/index.php/L-systems

Discussion and suggestion of further topics

Upcoming events

Next webinar

> The next webinar will be ~~Wednesday, April 28th, 2021 at 4 pm CET~~, **Wednesday, May 5th, 2021 at 4 pm CEST**

2nd international JSXGraph conference

> The 2nd international JSXGraph conference take place online **October 5th - 7th, 2021**. Stay tuned!