Workshop: Advanced JSXGraph

Vol. 6

Alfred Wassermann

UNIVERSITÄT BAYREUTH

5-05-2021

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/

JessieCode

What is JessieCode?

- The language JessieCode has been developed by Michael Gerhäuser, starting in 2011.
- JessieCode is open source, hosted at https://github.com/jsxgraph/JessieCode.
- License: LGPL and MIT, same as JSXGraph.

Motivation

- *Simplicity:* Avoid the use of JavaScript terminology for function plotting. Users can input `sin(x)` instead of `Math.sin(x)`, test it at
  - https://jsxgraph.org/wiki/index.php/Self-contained_function_plotting
  - https://jsxgraph.org/wiki/index.php/Even_simpler_function_plotter
- *Security:* In general, it should be avoided to allow users to input pure JavaScript code. There is the danger of XSS attacks. Filtering out dangerous expressions (like `eval`) is not good enough. JessieCode is a parsed language. It does not provide 100% security but is much more secure than allowing JavaScript input.
- *Usage scenario:* Allow students to input JessieCode in a forum discussion.
- *Extensibility:* Allow e.g. the development of symbolic math.

More details

- *Syntax:* The syntax of Jessiecode is close to JavaScript
- *Status:* While JessieCode is used widely for function plotting (probably without being recognized as JessieCode), the use as programming language for geometric / mathematical constructions is still not well known among JSXGraph users. Thus, there is not much feedback and their are certainly bugs.

JessieCode in JSXGraph elements

- JessieCode is automatically used if a string is supplied as argument of a function graph or a point.

```
var s = board.create('slider', [[0,3], [3,3], [0, 1, 5]], {name: 'S'});
var f = board.create('functiongraph', ["S.Value() * sin(x)"]);

var p = board.create('point', ['PI / 2', 'S'], {name: 'P'});
var q = board.create('point', ['PI / 2', 'Y(P) + 1'], {name: 'Q'});
```

- https://jsfiddle.net/n586pyco/3/
- Mathematical functions in JavaScript's `Math` object and in JSXGraph's `JXG.Math` can be accessed directly: Instead of `Math.sin(x)` one can use `"sin(x)"`.

one letter expressions like `x` may need to be changed to `1*x`.

- Access to slider values by name of the slider.
- Similarly, other elements can be accessed in JessieCode by their name (not their JessieCode variable name).
- Available functions:
  - `pow(b, e)`: `b` to the power of `e`, b^e
  - `log(x)`, `ln(x)`: natural logarithm
  - `log(x, b)`: logarithm to base b
  - `log2(x)`, `lb(x)`: logarithm to base 2
  - `log10(x)`, `ld(x)`: logarithm to base 10
  - `tan(x)`: tangent of x
  - `cot(x)`: cotangent of x
  - `cos(x)`: cosine of x
  - `cosh(x)`: hyperbolic cosine of x
  - `sin(x)`: sine of x
  - `sinh(x)`: hyperbolic sine of x
  - `acos(x)`: arccosine of x
  - `asin(x)`: arcsine
  - `atan(x)`: arctangent of x
  - `acot(x)`: arccotangent of x
  - `sqrt(x)`: square root of x
  - `cbrt(x)`: cube root of x
  - `nthroot(x)`: n-th root of x
  - `ratpow(x, m, n)`: rational power of x, $x^{m/n}$
  - `ceil(x)`: get smallest integer n with $n > x$.
  - `abs(x)`: absolute value of x
  - `max(a, b, c, ...)`: maximum value of all given values.
  - `min(a, b, c, ...)`: minimum value of all given values.
  - `exp(x)`: EULER to the x, e^x
  - `atan2(y, x)`: returns the arctangent of the quotient of its arguments.
  - `random(max = 1)`: generate a random number between 0 and max.
  - `round(v)`: returns the value of a number rounded to the nearest integer.
  - `floor(x)`: returns the biggest integer n with $n < x$.
  - `factorial(n)`: returns factorial $n!$
  - `trunc(v, p = 0)`: truncate v after the p-th decimal.
  - `V(s)`: returns the value of the given element, e.g. sliders and angles.
  - `L(s)`: returns the length of the given segment.
  - `X(P)`, `Y(P)`: returns the x resp. y coordinate of the given point.

  - `dist(P, Q)`: compute the distance of two points.
  - `deg(A, B, C)`: calculate the angle of three points in degrees.
  - `rad(A, B, C)`: calculate the angle of three points in radians.

JessieCode2JavaScript

JessieCode can be used in any JSXGraph / JavaScript program via `board.jc.snippet()`.

- If the second argument of this method is true, a function is created which evaluates and returns the first argument.
- The third argument contains the parameter(s) of the function, separated by commata, e.g. `'x'` or `'x,y'`.

```
var txt = 'x * sin(x^2)';
var f = board.jc.snippet(txt, true, 'x');
var plot = board.create('functiongraph', [f]);
```

- Inspect resulting JessieCode function: `console.log(f.toString());`
- Inspect resulting JavaScript function: `console.log(f.toJS());`

```
var txt = '2^(3*log2(4))';
var f = board.jc.snippet(txt, false);
console.log(f);
```

- https://jsfiddle.net/d1t2c6xe/
- See also https://jsxgraph.org/docs/symbols/JXG.JessieCode.html#snippet

JessieCode inside of `<script>`

JessieCode programs inside of HTML tags of the form

```
<script type="text/jessiecode">
</script>
```

will be interpreted and executed automatically. This means,

- A `<div>` element is generated and the JessieCode code is evaluated.
- Possible attributes for the `<script>` tag are: `width`, `height`, `boundingbox`, `container`, `axis`, `grid`
- Example: https://jsfiddle.net/nyt9hak7/1/

```
<script type="text/jessiecode" axis="true">
$board.setView([-10,4,10,-4]);

g = functiongraph(function(x) { return exp(x); });

s = slider([0,3], [5,3], [0,1,4]) <<name: 's'>>;

f = functiongraph(map (x) -> s * sin(x));
p1 = glider(PI, sin(PI), f) <<
    color: 'blue', size: 6, name: 'P', id: 'Pid'
  >>;

p1.moveTo([-2*PI, 0], 1000);
//P.move([-2*PI, 0], 1000);
//$('Pid').moveTo([-2*PI, 0], 1000);
</script>
```

- Another example: https://jsfiddle.net/jw1d6fy0/1/

```
<script type="text/jessiecode" axis="true">
for (i = 0; i < 100; i = i + 1) {
    point(10 * random()- 5, 10 * random()- 5) <<withLabel: false >>;
}

for (i = 0; i < 10; i = i + 1) {
    point(map (x) -> 10 * random()- 5, map (x) -> 10 * random()- 5) <<
       color: 'blue', withLabel: false >>;
}
// Attention: unused parameters in map are necessary
</script>
```

JessieCode has most elements of a programming language, i.e. variables, loops, functions, …

The *JessieCode language reference* below contains an overview of the language.

Symbolic math

- Example for symbolic derivative and algebraic simplification. (Definitely would need help from volunteers).
- See https://jsfiddle.net/wzvxqkfo/

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

<textarea id="input_code" cols="50" rows="10" style="float:left;">
f = map (x) -> x^2;
h = D(f, x);
```

```
</textarea>
<div style="width:120px; float: left; padding: 20px;">
  <p><button id="reset">Reset</button> </p>
  <p><button id="parse">Simplify code</button></p>
</div>
<textarea id="output_code" cols="80" rows="10" style="float:left;">
</textarea>
```

```
var a = board.create('slider',[[-4,4], [1,4], [1,2,4]], {name:'a'});
var f = board.create('functiongraph', ['a^x']);
var df = board.create('functiongraph', ['D(a^x, x)'], {strokeColor: 'red'
    });

parse = function () {
    return board.jc.manipulate(document.getElementById('input_code').value
        );
};

JXG.addEvent(document.getElementById('reset'), 'click', function () {
    document.getElementById('output_code').value = '';
}, this);
JXG.addEvent(document.getElementById('parse'), 'click', function () {
   document.getElementById('output_code').value = parse();
}, this);
```

Interplay JessieCode - JSXGraph

- See https://jsfiddle.net/d6wan15z/

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

// Set JessieCode variable `counter`
board.jc.parse("counter = 0;");

var p1 = board.create('point', [-2.0, 2.0]);

// Use `function() {return (counter < 5 || counter > 10) ? true: false; }`
var c1 = board.create('circle', [p1, 2.0], {
            visible: board.jc.snippet(
                "(counter < 5 || counter > 10) ? true: false", true,
                '')
        });

// Increase JessieCode variable `counter`
var button = board.create('button', [1, 4, 'increase counter',
    function() {
        board.jc.parse('counter = counter + 1;');
```

```
    }
]);

// Use function `function() {return counter; }`
var t = board.create('text', [2.5, 3,
            board.jc.snippet('counter', true, '')], {
            fontSize:24
        });
```

- https://jsxgraph.org/docs/symbols/JXG.JessieCode.html

JessieCode language reference

Data types

- **Boolean**, *true* or *false* (case insensitive, *tRuE* is a valid boolean constant).
- **Strings** are defined using single quote marks. Quote marks inside a string have to be escaped with a backslash.
- **Number**, corresponds to the JavaScript *number* data type.
- **Objects**, can be created only via object literal notation « » and the predefined element functions (see below). To access properties and methods the operator is used. Example:

```
obj = <<
  property: 'string',
  prop: 42,
  method: function (x) {
      return x*x;
  }
>>;
sixteen = obj.method(4);
```

- **Functions** are declared with the *function* operator

```
f = function (a, b, c) {
  return a + b + c;
};
```

- **Maps** are declared with the *map* operator

```
f = map (x) -> sin(x);
```

Comments

Only one line comments with `//` being the first non-whitespace characters are supported right now.

Operators

- **Logical operators**

Operator	Description
||	OR
&&	AND
!	NOT

- **Arithmetic operators**

Operator	Description
+	Addition
−	Subtraction or unary negation
*	Multiplication
/	Division
%	Modulus
^	Exponentiation

- **Assignment operators**

Operator	Description
=	Assignment

- **Comparison operators**

Operator	Description
==	Equals
<=	Less or equal
>=	Greater or equal
<	Less than
>	Greater than
!=	Not equal
~=	Approximately equal, can be used to compare two float values.

- **String operators**

Operator	Description
+	String concatenation

- **Member operators**

Operator	Description
.	Access the object's properties and methods

Control structures

The control structures are exactly the same as in JavaScript.

- **If**

```
if (<expression) {
  <Stmt>
} else if (<expression>) {
  <Stmt>
} else {
  <Stmt>
}
```

- **While loop**

```
while (<expression>) {
  <Stmt>
}
```

- **Do loop**

```
do {
  <Stmt>
} while (<expression>);
```

- **For loop**

```
for (<assignment>; <expression>; <assignment>) {
  <Stmt>
}
```

- **Predefined constants**

Name	Description
$board	Reference to the currently accessed board.
LN2	Natural logarithm of 2
LN10	Natural logarithm of 10
LOG2E	Base 2 logarithm of EULER
LOG10E	Base 10 logarithm of EULER
PI	Ratio of the circumference of a circle to its diameter
EULER	Euler's number e = 2.718281828459045
SQRT1_2	Square root of 1/2
SQRT2	Square root of 2

- **Predefined functions**: see above
- **$board methods**

Method	Description
`update()`	Update all dependencies and redraw the board.
`on(event, handler, context=board)`	Register an event handler for the given event.

Method	Description
`off(event, handler=)`	Deregister a given event handler or deregister all event handlers.
`setView(array, keepaspectratio=false)`	Changes the viewport. An array with 4 numbers is expected, the four numbers represent the left, upper, right and lower bound of the viewport. If keepaspectratio is true, the viewport is adjusted to the same aspect ratio as the board container.
`setBoundingbox(array, keepaspectratio=false)`	See setView.
`colorblind(type)`	Emulate color blindness. Possible types are *protanopia, tritanopia,* and *deuteranopia.*

- **Element functions**

Every element known to the loaded JSXGraph version is available inside JessieCode by its element type, e.g. points can be created by calling `point()`.

```
A = point(1, 2);
```

The given parameters correspond to the parents array of the `JXG.Board.create()` method. Attributes are given after the function call itself in an object:

```
A = point(1, 2) << strokeColor: 'red', face: '[]',
        size: 7, fillColor: 'black' >>;
```

For a complete list including documentation, see the JSXGraph docs.

Accessing elements

- **Variable assignment**

```
A = point(1, 2);
A.strokeColor = '#123456';
```

- **$**

```
point(1, 2) << id: 'foo', name: 'bar' >>;
$('foo').strokeColor = '#654321';
```

- **By id**

```
point(1, 2) << id: 'foo', name: 'bar' >>;
foo.strokeColor = '#f00f00';
```

This is possible only if **foo** is not used as a variable. This won't work:

```
foo = 1;
(function () {
  point(1, 2) << id: 'foo' >>;
  return foo.X();
})();
```

- **By name**

```
point(1, 2) <<
    id: 'foo', name: 'bar'
 >>;
bar.strokeColor = '#541541';
```

This is possible only if there is not a variable called **bar** in the current or any higher scope. See **By id** above for an example.

- **Setting attributes**

Attributes are set like object properties

```
A.size = 10;
A.face = '[]';
```

See the JSXGraph docs for available attributes. Texts and Points have two special attributes `X` and `Y` to set their coordinates.

- **Subelements**

Subelements like labels for points or the baseline in sliders or the dot indicating an angle element is a right angle can be accessed like properties

```
A.label.strokecolor = 'red';
```

The names used to access subelements correspond to their names used to set their attributes in `board.create()`.

- **Methods**

Not all methods of an element class are accessible in JessieCode. Currently these methods are available:

- all elements
  - setLabelText
- point
  - `move`
  - `moveTo`
  - `glide`
  - `free`
  - `X`
  - `Y`
- glider
  - all from point
  - `setPosition`
- text
  - `setText`
  - `free`
  - `move`
- slider
  - `Value`
- angle
  - `Value`

Upcoming events

Online Teacher Training Course on Programming JSXGraph, May 26th - June 9th, 2021

The course is intended to teachers and all interested people, who want to learn how to construct and program with the versatile JSXGraph library and use it in different online educational scenarios.

Requirements: None - this course is intended for non-programmers, there will be three online meetings.

The course is organized by the University of Maribor, Slovenia.

Free registration at https://docs.google.com/forms/d/e/1FAIpQLSdWB4f0tH4crHePFwzsm9QOlxEtsmi3RQsymo-Oo5LXPZ_duA/viewform

Next webinar

The next webinar will be **Wednesday, June 9th, 2021 at 4 pm CEST**

2nd international JSXGraph conference

The 2nd international JSXGraph conference take place online **October 5th - 7th, 2021**. Free registration at https://jsxgraph.org/conf2021.