# JSXGraph Workshop

International Meeting of the STACK Community 2025

Carsten Miller, Wigand Rathmann, Alfred Wassermann


8. April 2025, 12:00-13:00

# Contents

# 1 Preparation

## 1.1 Minimum requirements needed for JSXGraph

- A **web browser**: Firefox, Google Chrome, Microsoft Edge, Apple Safari. There is no big technical difference between these.
- **Internet connection**
- A **text editor**: (for offline development, only) We recommend *Microsoft Visual Studio Code* (its free), but there are countless other free alternatives. However, it should at least support *syntax highlighting*.
  - Download and install *Visual Studio Code* from https://code.visualstudio.com/download

## 1.2 Helpful but not mandatory

- Basic knowledge about **HTML**
- Basic knowledge about **JavaScript**

> Of course: the more, the better

## 1.3 More information for beginners

- JSXGraph introduction on youtube
- JSXGraph handbook
- JSXGraph wiki
- JSXGraph examples database
- API documentation

# 2 First steps with JSXGraph

## 2.1 Setup a web page

- **Online**: Open https://jsfiddle.net/k60dprL4/ in your web browser **or**
- **Offline**: Open a new file in your editor, paste the following code into it and open that file in your web browser.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>JSXGraph example</title>
    <link href="https://cdn.jsdelivr.net/npm/jsxgraph/distrib/jsxgraph.css" rel="
        stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph/distrib/jsxgraphcore.js"></
        script>
  </head>
  <body>

  <div id="jxgbox" class="jxgbox" style="width:600px; height:600px;"></div>

  <script>
  </script>

  </body>
</html>
```

Offline workflow: type text in editor – save file – reload web page in browser

## 2.2 Initialize JSXGraph

```
  <body>

  <div id="jxgbox" class="jxgbox" style="width:600px; height:600px;"></div>

  <script>
    var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-5, 6, 6, -5]});
  </script>

  </body>
```

- Book section

## 2.3 Our first geometry objects: points and lines

### 2.3.1 Creating points

```
board.create('point', [-2, 1]);
```

Another point:

```
board.create('point', [-2, 1]);
var q = board.create('point', [3, 0]);
```

- Book section

### 2.3.2 Creating lines

```
var line1 = board.create('line', [[-3, 1], [3, -1]]);

var p = board.create('point', [-2, -1]);
var q = board.create('point', [3, 1]);
var line2 = board.create('line', [p, q]);
```

- Book section

### 2.3.3 Add attributes to the objects

```
var p = board.create('point', [-2, -1], {name: 'first', size:5, color: 'blue'});
var q = board.create('point', [3, 1], {name: 'last', fixed:true, face: '[]'});

var line2 = board.create('line', [p, q], {straightLast:false, dash:4 });
```

- Book section on attributes
- Book section on circles

## 2.4 Function plotting

Again a template file for offline use:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>JSXGraph example</title>
    <link href="https://cdn.jsdelivr.net/npm/jsxgraph/distrib/jsxgraph.css" rel="
        stylesheet" type="text/css" />
    <script src="https://cdn.jsdelivr.net/npm/jsxgraph/distrib/jsxgraphcore.js"></
        script>
  </head>
  <body>

  <div id="jxgbox" class="jxgbox" style="width:600px; height:600px;"></div>

  <script>
    var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-5, 6, 6, -5], axis
        :true});
  </script>

  </body>
</html>
```

### 2.4.1 Static function graph

We have several possibilities to input a function term.

- Function graph given as string:

```
var funtxt = 'sin(x)';
board.create('functiongraph', [funtxt], {strokeColor: 'black'});
```

The content of the string is in the language *JessieCode* that has been developed specifically for JSXGraph. JessieCode is a full fledged programming language, but it's main purpose is to provide mathematical expressions and enable some symbolic manipulations. For more information, see the JessieCode github repository.

- Function graph given as JavaScript function:

```
board.create('functiongraph', [(x) => x * x], {strokeColor: 'red'});
```

In JavaScript, `(x)=> x * x` is called *arrow function*. It is easier to read than the old `function(x){ return x * x; }`.

Input of a function as string is convenient if the input comes from student. The JavaScript way to plot $\sin(x)$ would be `Math.sin(x)`, which is inaccessible to students. **Live example:** online function potter.

- See Book chapter on function graphs

### 2.4.2 Glider and tangent

Now, having a function graph, we can put a `glider` element onto it. This is a point that is bound to the function graph.

```
var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-5, 6, 6, -5], axis:
    true});
var funtxt = 'sin(x)';
var f = board.create('functiongraph', [funtxt], {strokeColor: 'black'});

var g = board.create('glider', [2, 0.5, f]);
```

Having this glider, we can add a tangent to function graph through this glider:

```
var t = board.create('tangent', [g]);
```

The complete example looks like this:

```
var board = JXG.JSXGraph.initBoard('jxgbox', {boundingbox: [-5, 6, 6, -5], axis:
    true});
var funtxt = 'sin(x)';
var f = board.create('functiongraph', [funtxt], {strokeColor: 'black'});

var g = board.create('glider', [2, 0.5, f]);
var t = board.create('tangent', [g]);
```

See it live at https://jsfiddle.net/1cpxnd96/.

# 3 STACK + JSXGraph

The idea is now to build a STACK question based on the example with the tangent at a graph. The question could be:

> Given is the function as shown in the diagram. Drag the point to a position where the function is stationary.

## 3.1 Get prepared

First, we will just generate a question containing the JSXGraph applet we wrote. A JSXGraph block starts with `[[jsxgraph width="500px"height="500px"]]` and ends with `[[/jsxgraph]]`. Between the delimiters, one can add the code. To initialize the board, the line now reads

```
var board = JXG.JSXGraph.initBoard(divid, {boundingbox: [-5, 6, 6, -5], axis:true})
    ;
```

`divid` contains the reference of the `div` layer generated for our board by `[[jsxgraph]]`.

## 3.2 Create the STACK question

Let's start by doing the following steps.

1. Create an empty STACK question
2. Name: Stationary point No question variables are needed yet.
3. Add to question text

```
[[jsxgraph  width="500px" height="500px" ]]
var board = JXG.JSXGraph.initBoard(divid, {boundingbox: [-5, 6, 6, -5], axis:true})
    ;
var funtxt = 'sin(x)';
var f = board.create('functiongraph', [funtxt], {strokeColor: 'black'});
```

```
var g = board.create('glider', [2, 0.5, f]);
var t = board.create('tangent', [g]);
[[/jsxgraph]]
```

4. Set the input type of `ans1` to 'Numerical', add a model answer to `ans1`, e.g., 1. Set 'Forbid float' to 'No'.
5. Fill `SAns` and `TAns` in node 1 of the PRT1 with 1.
6. Click on Preview.

### 3.3 Connect JSXGraph with STACK

Now the point can be dragged around, but no connection between the question variables and the JSXGraph part and back to `ans1` is established. Let's start with the connection between question variables and JSXGraph.

1. Define the function in the question variables and state a teacher's answer. We will even compute the first derivative of $f$

```
fun:sin(x);
dfundx:diff(fun,x);
tan1x:%pi/2;
tan1:float([tan1x,ev(fun,x=tan1x)]);
```

2. Change the model answer of `ans1` to `tan1`
3. Change node 1:

| field | value |
|---|---|
| Answert Test | NumAbsolute |
| SAns | `ev(dfundx,x=ans1[1])` |
| TAns | 0 |
| Test options | 0.1 |

Thus, the student can give the answer as a list, such as `[1.57,1]`. This prepares the binding of the glider `g` in the diagram with the student's answer `ans1`.

### 3.4 The binding

The next step is to glue the glider `g` to `ans1`. First, an option has to be added to assign something to be bound to `ans1`.

```
[[jsxgraph  width="500px" height="500px" input-ref-ans1="ans1Ref"]]
```

The binding of the glider `g` is done by the line

```
stack_jxg.bind_point(ans1Ref, g);
```

`stack_jxg.bind_point(ans1Ref, g);` establishes a bidirectional connection between `ans1` and `g`. When you have clicked the `Check` button, the student's answer will be reset in the diagram. Furthermore it allows to use the **Fill the correct responses** feature in the preview.

Binding in STACK docs on JSXGraph

JSXGraph Example in STACK docs

# 4 Various topics

## 4.1 "Dynamic" function graphs

In the next step we want to allow the students to manipulate the graph. For this we need a `slider` element.

```
var s = board.create('slider', [[-4, 5], [-1, 5], [-10, 1, 10]], {name: 'a'});
board.create('functiongraph', [
    (x) => s.Value() * x * x
], {strokeColor: 'red'});
```

- See Book chapter on sliders

## 4.2 Text elements – static and dynamic

```
board.create('text', [4, 3, 'Text'], {fontSize: 16});

board.create('text', [4, -3,
    () => 'f(2) = ' + s.Value() * 2 * 2
], {fontSize: 16});
```

JSXGraph texts also support MathJax and KaTeX.

## 4.3 Selected 3D examples

(From https://jsxgraph.org/share)

- 3D function graph with tangent plane

- Polyhedra
  - Cube with transformations
  - Subdivided icosahedron

### 4.4 JSXGraph moodle filter

There is also a way to display JSXGraph constructions *without* STACK. For this, the JSXGraph moodle filter has to be installed. A construction is added to a text element with the `<jsxgraph>` tag. Note that the ID of the hosting HTML tag is filled in automatically and replaces the user-specified `BOARDID`.

```
<jsxgraph>
var board = JXG.JSXGraph.initBoard(BOARDID);
board.create('polygon', [[-2,-2], [2,-3], [1,4]]);
</jsxgraph>
```

### 4.5 JSXGraph ILIAS page component

It is possible to use JSXGraph within ILIAS learning moduls. Surlabs maintaines the JSXGraph Page Component Plugin for ILIAS. The plugin comes with a code editor to simplify the embedding.

### 4.6 Example: How to bind a `point3d`

The function `stack` opens the way to bind a `point3d` e.g. to `ans1`.

```
[[jsxgraph  width="500px" height="500px" input-ref-ans1="ans1Ref"]]
"use strict";
/* transfer data */
var point1 = {#point1#};
//var point1 = [1,1,2];
console.log('point1=', point1);
var board = JXG.JSXGraph.initBoard(divid, {
    boundingbox: [-10, 10, 10, -10],
    axis: false,
    shownavigation: false
});
var box = [-2, 2],
    view = board.create("view3d", [
        [-6, -3],
        [8, 8],
        [box, box, box]
    ], { axesPosition: 'border' });
var point3d1 = view.create("point3d", point1, {
    fixed: true,
    name: 'Point 1'
}),
```

```
    point3d2 = view.create("point3d", [1.5, 1.5, 1.5], {
        name: 'Point 2'

    });
view.create("line3d", [point3d1, point3d2], { dash: 1 });
board.update();
/* Binding for point3d using custom binding */
let serializer = function () {
    return '[' + point3d2.X() + ',' + point3d2.Y() + ',' + point3d2.Z() + ']';
};
let deserializer = function (data) {
    let data2 = board.jc.snippet(data, false);
    point3d2.setPosition([data2[0], data2[1], data2[2]]);
};
stack_jxg.custom_bind(ans1Ref, serializer, deserializer, [point3d2]);
[[/jsxgraph]]
```

See custom binding in STACK Docs